METHOD AND APPARATUS FOR CONDUCTING DYNAMIC VOLUME TRACKING IN MOUNTABLE FILE SYSTEM

Publication number: JP3171238 (A)

Publication date: 1991-07-24

Inventor(s): BURAIAN EMU UIRUMAN; MAAKU JIEI ZUBIKOOSUKI; JIEIMUZU JII RETSUUIN; RAJIEN JIYAYANTEIRARU SHIYAA

Applicant(s): MICROSOFT CORP

Classification:

- international: ***G06F12/00; G06F15/16; G06F15/177; G06F17/30; G06F12/00;*** G06F15/16; G06F17/30; (IPC1-7): G06F12/00, G06F15/16

- European: G06F17/30F

Application number: JP19900227995 19900829

Priority number(s): US19890400531 19890829

Also published as:

JP7104808 (B)

JP2090000 (C)

EP0415346 (A2)

EP0415346 (A3)

EP0415346 (B1)

more >>

Abstract of **JP 3171238 (A)**

PURPOSE: To automatically map a file system in an indeterminate medium by comparing a volume identifier read from a medium with a identifier related to the file system. CONSTITUTION: The volume identifier is read from the medium by a loaded file system driver to be compared with the identifier related to the file system driver. Then when this identifier and the volume identifier coincide with each other, the file system driver is mounted. On the other hand when both identifiers do not coincide with each other, a next file system identified by the linked list of the file system driver is loaded. Thereby regardless of the form and the format of the file system the indeterminate medium is automatically and dynamically mapped to a proper file system.

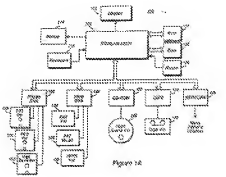

Figure 1d

⑲日本国特許庁（JP）　⑪特許出願公開

⑫公開特許公報（A）　平3－171238

⑤Int.Cl.[5]	識別記号	庁内整理番号	④公開　平成3年(1991)7月24日
G 06 F 12/00	301 B	8944－5B	
15/16	370 M	6945－5B	

審査請求　未請求　請求項の数　1　（全39頁）

⑤発明の名称　設置可能なファイルシステムにおいてダイナミックボリュームトラッキングを行う方法及び装置

⑳特　願　平2－227905
㉒出　願　平2(1990)8月29日
優先権主張　㉜1989年8月29日㉝米国(US)㉞400531

㉗発明者　ブライアン　エム　ウイルマン　アメリカ合衆国　ワシントン州　98007　ベルヴィュー　1308　エヌ　ノースイースト　フォーティサード　プレイス　14545
㉘出願人　マイクロソフト　コーポレーション　アメリカ合衆国　ワシントン州　98052－6399　レッドモンド　ワン　マイクロソフト　ウエイ（番地なし）
㉙代理人　弁理士　中村　稔　外7名

最終頁に続く

明　細　書

1.発明の名称

設置可能なファイルシステムにおいてダイナミックボリュームトラッキングを行う方法及び装置

2.特許請求の範囲

データ記憶装置とコンピュータシステムとの通信に使用するファイルシステムをマウントする方法において、

a）前記ファイルシステムがリンクされたシーケンスに編成されている省略時ファイルシステムを備えたコンピュータオペレーティングシステムに複数のファイルシステムモジュールを設け、

b）前記コンピュータシステムにデータ記憶装置を接続し、

c）前記コンピュータシステムが前記データ記憶装置にアクセスする最初の時点で、前記データ記憶装置におけるメディアの変化を検出し、

d）前記ファイルシステムのリストにおいて識別されたファイルシステムをローディングし、

e）前記メディアから、ローディングされた前記ファイルシステムにより位置が特定化されたボリューム識別子を読み取り、

f）前記メディアから読み取られた前記ボリューム識別子を、前記ファイルシステムに関連する識別子と比較し、

g）前記両識別子が一致する場合には、前記ファイルシステムをマウントし、

h）前記両識別子が一致しない場合には、ファイルシステムの前記リストにおいて識別された次のファイルシステムをローディングし、

i）ファイルシステムの前記リストにおける各ファイルシステムが試験されるまで又は一致が見出されるまで、前記工程（e）に戻り、

j）一致が全く見出されない場合には省略時ファイルシステムをマウントすることを特徴とするデータ記憶装置とコンピュータシステムとの通信に使用するファイルシステムをマウントする方法。

3.発明の詳細な説明

本明細書には、３８５個のフレームを含むマイクロフィッシュの４枚のシートからなる付録Ｉ（Appendix I）が含まれている。

本発明はコンピュータ制御システムの分野に関し、より詳しくは、コンピュータシステムを備えた装置同士の通信を行う方法及び手段に関する。

一般に、コンピュータシステムは、中央処理装置と、ランダムアクセスメモリと、リードオンリメモリと、データ入力装置、データ出力装置、フロッピディスク及び固定ディスク又はハードディスク等の種々の不揮発性データ記憶装置等の種々の周辺装置とを有している。一般に、それぞれの装置間の通信はコンピュータオペレーティングシステムにより制御される。良く知られた１つのコンピュータオペレーティングシステムとして、マイクロソフト社（Microsoft）から市販されているMS-DOSオペレーティングシステムがある。

MS-DOSオペレーティングシステムにおいては、単一のファイルシステムが、周辺装置に記憶されたファイルの編成を記載しかつ構成している。コンピュータシステム及びそれぞれの周辺装置の両者により認識されたフォーマット中のデータをコンピュータシステムが読み取り又は書き込みできるようにするには、データはこのファイルシステムに従って編成されなくてはならない。例えば、MS-DOSオペレーティングシステムに使用される従来のフロッピディスクを用いた周辺装置においては、フロッピディスクのデータは、FAT ファイルシステム（ファイル割当てテーブル（file allocation table）を用いていることから、このように命名されている）として知られているファイルシステムに従って構成されている。FAT ファイルシステムは、今日、世界中で最も広範囲に使用されているファイルシステムの１つである。テープ記憶装置のような、周辺装置の他の形式のデータ記憶装置には、他のファイルシステムを接続することもできる。

ファイルシステムにより、オペレーティングシステムのカーネルとデバイス従属ドライバ（device dependant drivers）との間の通信を容易に行うことができる。また、ファイルシステムは、オペレーティングシステムのカーネルにより発せられた読取り及び書込みコマンド（並びに、ファイルを開閉する機能）を、デバイスドライバが認識できるフォームに変換することに必要とすることができる。

MS-DOSオペレーティングシステムを用いる場合には、オペレーティングシステムは、コンピュータシステムに用いられている特定の周辺装置に使用できる適合ファイルシステムを構成しなければならない。一旦ファイルシステムが構成されたならば、このファイルシステムは、オペレーティングシステムが変更されない限りそのままであり、すなわち変更されることはない。このため、一般に、広範囲のプログラム作成努力と多大の消費時間とが要求される。また、コンピュータオペレーティングシステムについての広範囲の知識が必要であり、オペレーティングシステムの詳細にアクセスできない人は、ファイルシステムを容易に変更することはできない。

また、従来のシステムにおいては、異種ファイルシステム（foreign file systems）のファイルを収容しているディスクメディアを、固有システム（native system）に使用することはできない。例えば、多数の製造業者（各製造業者は別々のファイルシステム構成に継続している）により多くのコンピュータシステムが多年に亘って開発されている。現在のスタティックファイルシステムの技術では、一般に、或る１つのシステムからのディスクメディアを別の形式のシステムで機能させることはできない。コンピュータは一層ポピュラーなものとなっているため、あらゆる形式のコンピュータシステムの間でファイルを共用できるようにすることの重要性が増大している。事実上知られているあらゆるコンピュータシステムからのディスクメディアを、単一のオペレーティング環境において自動的に認識しかつ読み取ることができるシステムは未だ存在しない。また、コンピュータオペレーティングシステムのカーネルを変え

も必要なくして、或るシステムに付加（又は変更）できるファイルシステムは未だ存在しない。

簡単に云えば、本発明は、コンピュータシステムに使用されるメディアを自動的に識別しかつ前記メディアを認識するファイルシステムを自動的かつダイナミックにマウントできる方法及び手段に関するものである。本発明の好ましい実施例によれば、省略時ファイルシステム（default file system）を備えていて、リンクされたシーケンスに従ってファイルシステムを編成するように構成されたコンピュータシステムに、1つ以上のデータ記憶装置及び複数のファイルシステムドライバを設けることができる。このコンピュータシステムは、該コンピュータシステムの全ての周辺装置を連続的にモニタリングしていて、周辺記憶装置におけるメディアのあらゆる変化を検出するようになっている。データ記憶装置のメディアが変更されるあらゆる場合、又は最初にコンピュータシステムがデータ記憶装置にアクセスする場合には、ファイルシステムドライバのリストにおいて識別された最初のファイルシステムドライバがローディングされ、ローディングされたファイルシステムドライバによりボリューム識別子（volume identifier）の位置が特定化されているメディアからボリューム識別子が読み取られる。メディアから読み取られたこのボリューム識別子は、次に、ファイルシステムドライバに関連している識別子と比較され、この識別子とボリューム識別子とが一致するときにはファイルシステムドライバがマウントされる。両識別子が一致しない場合には、ファイルシステムドライバのリンクされたリストにおいて識別された次のファイルシステムドライバがローディングされる。

その後、このプロセスは、ファイルシステムドライバのリンクされたリストにおける各ファイルシステムドライバが試験されるまで、又は一致が見出されるまで繰り返される。一致が見出されない場合には、省略時ファイルシステムがマウントされる。

従って本発明の目的は、コンピュータシステムが、多数のメディア形式の全てを識別できるようにし、かつ適正なファイルシステムをマウントして前記メディアに使用できるようにする方法及び手段を提供することにある。

本発明の他の目的は、不確実メディアにファイルシステムを自動的にマッピング（配置）する方法及び手段を提供することにある。

本発明の他の目的は、ユーザからのインタラクションなくして、自動的に不確実メディアに適用できるコンピュータシステムを提供することにある。

本発明の他の目的は、オペレーティングシステムのカーネルを変更する必要なくして、ファイルシステムを変更でき又はコンピュータオペレーティングシステムに付加できるようにコンピュータオペレーティングシステムを改善することにある。

本発明の他の目的は、メディアのフォーマットのあらゆる従属性が適当なファイルシステム内に密閉（encapsulated）されている不確実なメディアにファイルシステムを自動的にマッピングできる方法及び手段を提供することにある。

本発明の更に他の目的は、任意に設置できるファイルシステムをコンピュータシステムに設けることができるようにする方法及び手段を提供することにある。

本発明のこれらの目的及び他の目的は、添付図面を参照して以下に述べる本発明の詳細な説明により明らかになるであろう。

第1図には、本発明の原理に従って構成されたコンピュータシステム100が示されている。このコンピュータシステム100は、中央処理装置すなわちマイクロプロセッサ102と、ランダムアクセスメモリ104と、リードオンリメモリ106と、マウス108及びキーボード110のような入力装置と、ディスプレイ112及びプリンタ114のような出力装置と、フロッピディスクドライブ116、ハードディスクドライブ120、CD-ROMドライブ122及びテープドライブ124等からなる種々の不揮発性記憶装置とを有している。また、このコンピュータシステム100は、

ネットワーク１３６と通信できるようになっている。不揮発性記憶とは、装置の電源を遮断してもデータが消去されないことをいう。

従来のシステムにおいては、オペレーティングシステムは、各周辺装置が単一のメディア形ファイルシステムドライバのみと互換性をもつファイルシステムドライバによりスタティックに構成されている。指定のファイルシステムドライブとの互換性のないドライブにメディアが供給されると、メディアは首尾良くアクセスすることができない。以下に説明するように、本発明は、周辺装置とは独立してかつメディアに関するデータのフォーマット又は位置についての条件を賦課することなくして、関連するファイルシステムにメディアを自動的にマッピングする方法及び手段を提供するものである。例えば、フロッピドライブユニット（フロッピディスクドライブ）１１６は、多数のファイルシステムに従ってフォーマット化されたボリューム（例えば、FAT ファイルシステムに従ってフォーマット化されたボリューム１２８、良く知られたHigh Sierra ファイルシステムに従ってフォーマット化されたボリューム１３２、及びもう１つのファイルシステムに従ってフォーマット化されたボリューム１３０）に使用することができる。同様に、ハードディスク（ハードディスクドライブ）１２０の様々の区分（パーティション）は、ボリューム１３４、１３６、１３８として表した多数のファイルシステムに従ってフォーマット化することができる。同様に、CD-ROMドライブ１２２及びテープシステム（テープドライブ）１２４は、ボリューム１４０、１４２（これらは、それぞれのファイルシステムに従ってフォーマット化されている）に使用することができる。また、ネットワーク１２６は、サーバ（該サーバは、それら自体のファイルシステムに従って作動する）を備えた任意の数のネットワークに接続することができる。

コンピュータシステム１００の作動（オペレーション）は、良く知られた多数のオペレーティングシステムのうちの任意のオペレーティングシステムにより調整することができる。しかしながら、本発明は特に、Microsoft 社により開発されたOS/2オペレーティングシステムに使用するのに適している。本発明の作動環境（operating environment）の構成が第１８図に示してある。一般に、アプリケーション１５２は、カーネル１５４により処理されるファイルシステムのリクエストを発生する。次いで、カーネル１５４は、このリクエストを適当なファイルシステムドライバ（FSD）１５６～１７８に導く。任意のファイルシステムドライバを、多数のハードウェア装置と協働させることができる。例えば、ボリューム172 、174 についてファイルシステム作動を行う場合には、High Sierra ファイルシステム１５６をCD-ROMプレーヤ（CD-ROM ドライブ）１２２及びディスクドライブ１１６に使用することができる。同様に、FAT ファイルシステム１６０及びHPFSファイルシステム１６２の両者は、ボリューム176 、178 （これらの各々は、ハードディスク１２０にある）についてのファイルシステム作動を行うのに使用することができる。また、ボリューム１８０についてファイルシステム作動を行う場合には、ディスクドライブ１１６にファイルシステムドライバを使用することができる。従って、本発明によれば、ファイルシステムの形式及びフォーマットの如何に係わらず、適当なファイルシステムに不確実メディアを自動的にかつダイナミックにマッピングする方法及び手段が提供される。

第２Ａ図は、従来技術によるMS-DOSオペレーティングシステムのファイルシステム構成を示すものである。MS-DOSオペレーティングシステム200 においては、オペレーティングシステムのカーネル２０４内にFAT ファイルシステム２０２が埋設されている。このFAT ファイルシステム２０２はオペレーティングシステムのカーネル２０４内に一体化されているため、変更（モディファイ）することは困難である。また、付加的なファイルシステムが必要な場合には、オペレーティングシステムのカーネル２０４を書き換えてそれらのファイルシステムに適合できるようにしなければなら

ない。

本発明によれば、第２Ｂ図に示すシステム技術により上記問題点を解決することができる。本発明のコンピュータシステム１００においても、OS/2カーネル２５２内には、FAT ファイルシステム２０２が搭載されている。しかしながら、本発明によれば、オペレーティングシステムのカーネル２５２に対して外部装置であるFAT ファイルシステムドライバ２５４、２５６、２５８をダイナミックに取り付ける方法及び手段が提供される。即ち、設置可能な３つのファイルシステムドライバを備えたシステム２５０が示されているが、実際には、本発明は、ファイルシステムドライバの数に制限されることはない。

設置可能（installable）なファイルシステムドライバ（file system driver、「FSD」）は、多くの点でデバイスドライバに類似している。FSDは、ダイナミックリンクライブラリ（dynamic-link library、「DLL」、一般には、SYS 又はIFSエクステンションを備えている）のように構成されたファイルのディスク上に存在し、CONFIG.SYSファイルにおけるIFS＝ステートメント（statements）によるシステムの初期化中にローディングされる。IFS＝宣言（directives）は、それらが出合う命令において処理され、また、デバイスドライバについてのDEVICE＝文（statements）の命令に対して類似する。これにより、ユーザが、非標準デバイス用のデバイスドライバをローディングし、該デバイスのボリュームからファイルシステムドライバをローディングすること等が可能になる。一旦FSD が設置されかつ初期化されると、カーネルは、ファイルの開放、読取り、書込み、シーク（seeks）、閉鎖等についての論理的リクエスト（logical request）の言語で、FSD と通信する。FSD は、ボリューム自体に関連される関連構成及びテーブルを用いて、これらのリクエストをセクタ読取り（sector reads）用のリクエストに翻訳し、かつ、ファイルシステムヘルパ（File System Helpers、「FsHlps」）と呼ばれる特別なカーネル入口点を呼び出すことができる書込みを行う。カーネルは、セクタI/O に対するデマンドを適当なデバイスドライバに導き、かつその結果をFSD に戻す。

ボリュームをFSDs（複数のFSD）と結合させるべくオペレーティングシステムにより用いられる手順は、ダイナミックボリュームマウンティング（dynamic volume mounting）と呼ばれ、次のように作動する。ボリュームが最初にアクセスされるとき、或いは、直接アクセスを行うべく（例えばFORMATオペレーションにより）ボリュームがロックされ次いでアンロックされた後に、オペレーティングシステムのカーネルは、ボリュームからFSDsの各々への識別情報を発生し、これは、FSDがこの情報を認識するまで順次行われる。FSD がボリュームをクレイムすると、ボリュームがマウントされ、ボリュームに対する全ての連続ファイルI/O リクエストが、ボリュームをクレイムしたFSD に導かれる。

この構成により、従来技術にはない幾つかの利点を得ることができる。例えば、不確実メディアがコンピュータシステムに与えられる場合に、コンピュータシステムは、利用できるファイルシステムドライバを走査して、このメディアを認識できるファイルシステムドライバを位置付けすることができ、これにより、メディアへのファイルシステムドライバの自動マッピングを行うことが可能になる。また、オペレーティングシステムのカーネルを変更する必要なくして、ファイルシステムドライバを更新することができる。更に、新しい形式の周辺装置が開発されたときに、既存システムのソフトウェアを混乱させることなく、適当なファイルシステムドライバをオペレーティングシステムに付加することができる。

コンピュータシステム１００のより詳細なダイアグラムが第３図に示してある。コンピュータシステム１００は、アプリケーションプログラム３０２とディスク装置３０４のようなデータ記憶装置との間の通信を行うことができるオペレーティングシステムのカーネル２５２を有している。また、このコンピュータシステム１００は、ファ

イルシステムドライバ254～258と関連して作動するデバイスドライバ306を有している。図面には単一の周辺装置304を備えたコンピュータシステム100が示されているが、本発明は、任意の数の論理的又は物理的周辺装置と組み合わせて使用できるものである。

作動に際し、アプリケーションプログラム302は、所望の機能についての入口点を呼び出すことにより、オペレーティングシステムのカーネル252に対する論理的ファイルリクエストを発行する。これらの機能には、ファイルを開放すること（DosOpen）、ファイルを読み取ること（DosRead）、ファイルに書き込むこと（DosWrite）等のリクエストを含めることができる。オペレーティングシステムのカーネル252は、これらのリクエストを、ファイルを保持（ホールディング）する特定のボリュームについての適当なファイルシステムドライバ254～258に導く。次に、適当な設置可能なファイルシステムドライバが、論理的ファイルリクエストを、指定メディアの論理的セクタの読取り及び書込みのためのリクエストに翻訳し、かつオペレーティングシステムのカーネルのファイルシステムヘルパ308を呼び出して、これらのリクエストを適当なデバイスドライバ306に導く。ファイルシステムヘルパ（FsHlps）308については、以下により詳細に説明する。デバイスドライバ306は、オペレーティングシステムのカーネルからの論理的セクタリクエストを、特定の物理ユニット（すなわち、メディアのシリンダ、ヘッド及びセクタ）についてのリクエストに変形し、かつ、ディスク装置にコマンドを発行して、ディスクメディアとランダムアクセスメモリ210との間にデータを伝達する。

次に、物理装置を特定のファイルシステムにマッピングすることについて以下に詳細に説明する。MS-DOS環境（MS-DOS environment）においては、フロッピディスクはボリュームと呼ばれる。固定ディスク（又はハードディスク）は、多数のボリュームに区分することができる。このターミノロジが、本発明に首尾良く適用されている。簡単に云えば、コンピュータシステムが最初にブート（boot）されるとき、ボリュームが最初にアクセスされるとき、又はコンピュータシステムが、ディスク装置304内に不確実メディアが存在することを決定するときにはいつでも、コンピュータシステムは、ファイルシステムドライバのリンクされたリストにおける最初のファイルシステムドライバを試験する。ファイルシステムドライバがディスク装置にローディングされたボリュームを認識する場合には、ファイルシステムドライバがマウントされる。そうでない場合には、コンピュータシステムは、メディアを認識するファイルシステムドライバが位置付けされるまで、利用できるファイルシステムドライバを連続的にポーリングする。関心をもつメディアを認識する設置可能なファイルシステムドライバが全く見出されない場合には、省略時ファイルシステムドライバがマウントされる。本発明の好ましい実施例においては、省略時ファイルシステムは、上記のFAT ファイルシステムである。

不確実メディアは、幾つかの方法により検出することができる。ディスク装置には機械的なラッチ機構が設けられており、該ラッチ機構は、ディスクがディスク装置から取り出されるとき又はディスク装置に装填されるときに作動する。一般にラッチ機構は、ドライブの次の作動によりドアが開放されたことを示すように機能する。デバイスドライバがこの表示を受け取ると、エラー不確実メディア（ERROR_UNCERTAIN_MEDIA）がオペレーティングシステムに戻される。機械的なラッチ機構がないシステムにおいては、所定時間より短い時間内にメディアを変更できないと考えられる。本発明の好ましい実施例においては、この時間は2秒であると考えられる。従って、所定時間以上の時間をかけても特定のボリュームがアクセスされない場合には、このメディアは不確実であると推定される。

第4図は、FAT ファイルシステムのディスクフォーマットのダイアグラムである。FAT ファイルシステムは、MS-DOSオペレーティングシステムの

初期から該MS-DOSオペレーティングシステムに使用されている。FAT ファイルシステムについての詳細な説明が、Duncan著、「アドバンスMS DOS プログラミング("Advance MS DOS Programming")」(Microsoft Press 社刊、1986、1988) においてなされている。FAT ファイルシステムは、FAT ファイルシステムは、ファイル割当てテーブル (File Allocation Table) を中心題目としている。各論理的ボリュームはそれ自体のFAT と関連していて、2つの重要な機能を有している。すなわち、各論理的ボリュームは、割当てユニット (allocation units) のリンクされたリストの形態をなすボリュームに関する各ファイルについての割当て情報 (allocation information) を収容していて、その割当てユニットには、割当 (又は拡大) されているファイルへの割当て (代入、assignment) がないことを表示する。

FAT ファイルシステムに従ってディスクがフォーマット化されると、ブートセクタ (boot sector) がセクタゼロと書き込まれる。ファイル割当てテーブルの後にはルートディレクトリ(root directory) が続き、このルートディレクトリの後にはボリュームファイルが続く。ブートセクタには、ブートパラメータブロックすなわちBPB と呼ばれる。或る領域におけるボリュームに関する種々の記述情報(descriptive information) 、ドライブの数及びボリュームI.D.のような情報、及びブートストラップルーチン(bootstrap routine) が収容されている。

ファイル割当てテーブルは、ディスクについての割当て可能なクラスタ (これらのクラスタは、セクタを2乗したものである) に直接相当するフィールド (欄) に区分される。一般に、これらのフィールドは16ビットの幅を有している。最初にリザーブされたFAT 入口には、BPB においても見出すことができるメディア記述バイト (media descriptor byte)のコピーが収容されている。リザーブされた残余のフィールドにはOFFHが収容されている。残余のFAT 入口には、それらの相当ディスククラスタ(corresponding disk clusters) の使用が記述される。ディレクトリの各ファイルの入口には、これらのファイル (該ファイルは、FAT への入口点として使用される) に割り当てられる最初のクラスタの数が収容される。入口点から、各FAT スロット (FAT slot) は、最終クラスタマークに出合うまで、ファイル内の次のクラスタの番号を収容する。また、FAT ファイルシステムには、読取りエラー等によるFAT のセクタへのアクセスが失敗した場合に用いることができる最初のファイル割当てテーブルの複製を維持する機能をオプションとして設けることができる。

ファイル割当てテーブルの後には、ルートディレクトリが続く。このルートディレクトリは、ファイル、他のディレクトリ、及びオプションとしてのボリュームラベルを記述する32バイトの入口を収容している。

ルートディレクトリの後の残余のボリュームは、クラスタのプールとして見ることができるファイル領域 (各ファイル領域には1つ以上の論理的セクタが収容されている) として知られている。各クラスタは、FAT の対応入口 (該入口には、FAT の現在使用、すなわち、利用できること、リザーブされていること、ファイルに割り当てられていること、又は使用できないことが記述されている) を備えている。

FAT ファイルシステムは、1Mb以下のボリュームで優れた性能を得ることができる。しかしながら、ボリュームのサイズが1Mbを超えると、FAT ファイルシステムの性能は急激に低下する。容易に入手可能なハードディスクのサイズは急激に増大しているため、このことは重要な問題となっている。

ボリュームが1Mb以下の場合には、FAT は、いつでもランダムアクセスメモリ内に保持される程充分に小さく、従って、ファイルのいかなる部分にも非常に高速のランダムアクセスを行うことができる。しかしながら、ハードディスク又は固定ディスクに適用した場合には、FAT は大き過ぎてメモリに保持できなくなり、かつ細分してメモリ内にページ付けしなければならない。このため、

多くの余分なディスクヘッド運動が必要になり、コンピュータシステムのスループットを低下させている。また、ディスクの空きスペース(free space)についての情報が、FATの多数のセクタを横切って分散されるため、連続的にファイルスペースを割り当てることは実際的でない。このため:ファイルが細分化され、コンピュータシステムのスループットが更に低下される。また、ハードディスクに比較的大型のクラスタを用いるため、無駄なスペースが非常に大きくなる。

第5A図～第5H図には、設置可能なファイルシステムの1つの場合のディスクフォーマットを示す一連のダイアグラムが示されている。このファイルシステムは、高性能ファイルシステム(High performance file system、HPFS)と呼ばれているものである。本発明の高性能ファイルシステムは、FATファイルシステムについての上記問題を解消でき、かつあらゆる形式のディスクメディアについて優れた性能を発揮できるものである。第5A図に示すように、HPFSボリュームは、前に形成されたFATパーティション形の領域に沿って固定ディスク上に設けることができる。HPFSボリュームは、512バイトのセクタサイズを使用しており、2199Gb(2^{32}のセクタ)の最大サイズを有している。HPFSは固定ディスクに使用することを主として設計されているが、実際上、あらゆる形式のディスクメディアとの互換性を有している。

HPFSボリュームは、固定された構成が始ど必要とされない。ブートブロック502にはボリューム(8Mb)のセクタ0～15が割り当てられ、該セクタ0～15は、ボリュームのネームフィールド(名前欄)504、32ビットボリュームのIDフィールド506、BIOSパラメータブロック508、ディスクブートストラッププログラム510を収容している。ディスクブートストラッププログラム510は、オペレーティングシステムファイルが見出される限りは、これらのオペレーティングシステムファイルの位置付け及び読取りを行う限定モードで使用することができる。

ブートブロック(BootBlock)502の後には、スーパーブロック(SuperBlock)512及びスペアブロック(SpareBlock)514が続いている。スーパーブロック512は、ディスクメインテナンスユーティリティによって変更されるに過ぎない。スーパーブロック512は、空きスペースのビットマップを指すポインタ516、バッドブロックリスト518、ディレクトリブロックバンドを指すポインタ520、ルートディレクトリを指すポインタ522を収容している。更にスーパーブロック512は、デート(日付け)を備えたデートフィールド(日付け欄)524を収容しており、ボリュームは、CHKDSKにより最終チェック及び修復がなされる。CHKDSKは、ディスクの悪い部分を検出しかつカタログするための良く知られたOS/2ディスクユーティリティである。

スペアブロック514は種々のフラグ及びポインタを収容しており、これらについては以下に詳述する。スペアブロック514は、コンピュータシステムが実行されるときに変更される。

残余のボリュームは、ファイルの記憶に使用される8Mbバンド(例えば、バンド516～522)に区分される。第5A図には4つの8Mbバンドが示されているが、HPFSは非常に多数のバンドを得ることができる。各バンドには、それ自体の空きスペースビットマップ(例えば、ビットマップ524～534参照)が設けられている。空きスペースビットマップの各ビットは、セクタを表している。セクタが使用されている場合にはビットは0であり、セクタが使用可能(アベイラブル)であるときは、ビットは1である。ビットマップは、バンドのヘッド又はテールに位置付けされるため、2つのビットマップは交互のバンドの間に隣接している。ビットマップのバンドサイズは、任意のサイズのファイルを収容できるように変更できるけれども、上記構成により、16Mbになるようにファイルに割り当てることができる最大の連続空きスペースを得ることができる。ディスクのシークセンタにおいて(又はシークセンタに向かって)位置付けされた1つのバンドは、ディレ

クトリブロックバンドと呼ばれ、後述するような特別の処理を受ける。

HPFSの全てのファイル又はディレクトリは、第５Ｂ図及び第５Ｃ図に示すFnodeと呼ばれている基本ファイルシステムの目的（オブジェクト）にアンカーされている。Fnode５３０は、ファイル又はディレクトリに割り当てられた最初のセクタすなわち第１セクタであり、スーパーブロック５０４におけるフィールド５２２により指示されている。各Fnode５３０は単一のセクタを占領し、かつ、第５Ｂ図に示すように、ファイルシステムにより内的に用いられる識別及びアクセス情報フィールド５４０、拡大属性（extended attribute、「EA」）及びアクセス制御リスト(access control lists、「ACLs」）を記憶する領域５４２、関連するファイル又はディレクトリのネームの長さ及び最初の１６文字を表示したい場合にはそのためのフィールド５４４、及び割当て構成５４６を収容している。Fnodeは、これを代表するファイル又はディレクトリの近くに常に記憶されている。

第５Ｃ図に示す割当て構成５４６は、ファイル又はディレクトリの連続性のサイズ及び度合に基づいて幾つかの形態をとる。本発明のHPFSは、１つ以上の連続セクタの１つ以上の実行（runs）又はエクステントの割当てとして、ファイルをビュー（views）している。各実行は、１対の二重ワード、すなわち、セクタにおける３２ビットのスタートセクタ数及び３２ビットの長さ（この長さは、実行長さエンコーディングと呼ばれている）により記号化される。アプリケーションプログラムの観点からすると、エクステントは目で見ることはできない。すなわち、ファイルはバイトの継ぎ目のない流れであると考えることができる。

Fnodeにおける割当て情報にリザーブされたスペースは、各１６Mbまでのセクタの８個の実行と同数のポインタを保持することができる。従って、高度連続サイズのかなり小さなファイルを、Fnodeの中に完全に記述することができる。

HPFSは、Fnodeにとっては大き過ぎるか細分され過ぎているファイルの位置を示す新しい方法を採用しており、８個以上の実行を有している。Fnodeの割当ては、割当てセクタのB＋ツリー（木）のルート（根）となり、このルートには、第５Ｄ図に示すように、ファイルのセクタ実行を指す実際のポインタが収容されている。B＋ツリー及びB－ツリーの概念については後で詳述する。Fnodeのルートは、１２のエレメントのためのルームを有している。各割当てセクタは、種々のセクタ情報に加えて、セクタ実行を指す４０個程のポインタを収容することができる。従って、本発明の好ましい実施例においては、２レベル割当てのB＋ツリー（two level allocation B＋ Tree)が、7.58 Gb（１２＊４０＊１６Mb）の理論的最大サイズをもつ４８０（１２＊４０）個のセクタ実行のファイルを記述することができる。

これとは異なり、高度に細分化されたファイルを記述するには２レベル割当てのB＋ツリーが充分でない場合には、HPFSファイルシステムが、ツリーに必要なだけの付加的レベルを導入する。中間レベルにおける割当てセクタは、６０個程の内部（端部ではない）B＋ツリーノードを保持することができ、このことは、この構成の記述能力が極めて大きな数に急速に成長することを意味している。例えば、３レベル割当てB＋ツリーは、28,800（１２＊６０＊４０）個のセクタ実行を記述することができる。

割当てセクタの実行長さエンコーディング(run-length encoding)及びB＋ツリーは、ファイルのサイズ及び位置を充分に特定できるメモリであり、かつ従来技術に比べ幾つかの優れた長所を有している。セクタ数への論理的ファイルオフセットの翻訳は極めて高速に行われる。すなわち、ファイルシステムは、正しい範囲が見出されるまで実行サイズを要約し、実行ポインタ(run pointers)のリスト（すなわちリストのB＋ツリー）を単に横切るだけである。そのとき、簡単な計算を行うことにより、実行の中でセクタを識別することができる。また、新たに割り当てられたセクタがファイルの前の最終セクタと連続している場合には、実行長さエンコーディングによって、ファ

イルを論理的に拡大することが極めて簡単になる。このファイルシステムは、ファイルの最終実行ポインタのサイズ二重ワード（size double-word）を単に増大させるだけであり、適当な空きスペースのビットマップにおけるセクタのビットをクリアするように構成されている。

ファイルと同様に、ディレクトリはFnodesにアンカーされる。ルートディレクトリ（root directory）についてのFnode を指すポインタは、スーパーブロック５１２において見出すことができる。第５Ｂ図は、本発明によるディレクトリ構成を示すものであり、ここにはディレクトリFnode 550が示されている。ルート以外のディレクトリについてのFnode は、それらの親ディレクトリ（parent directories）におけるサブディレクトリ入口を通って到達する。

ディレクトリは、ディスク上に４つの連続セクタ（consecutive sectors）として割り当てられる２Kbのディレクトリブロックから構成されていて、任意のサイズに成長することができる。例えば、ディレクトリブロック５５２、５５４、５５６を参照されたい。このファイルシステムは、ディスクのシークセンク又はその近くに位置付けされたディレクトリバンドにディレクトリブロックを割り当てることを試みている。ディレクトリバンドが満たされると、スペースが利用できる限り、ディレクトリブロックが割り当てられる。

２Kbの各ディレクトリブロックには、１つから多数のディレクトリ入口を設けることができる。例えば、入口５５８～５６８を参照されたい。ディレクトリ入口には幾つかのフィールド（欄）が設けられ、これらのフィールドには、第５Ｂ図に示すように、時間及び日付スタンプのためのフィールド５７０と、Fnode ポインタを収容するフィールド５７２と、ディスクメインテナンスプログラム（このプログラムは良く知られたものである）による使用ができるようにするための用法カウントフィールド（usage count field）５７４と、ファイル長さすなわちディレクトリネームを収容するフィールド５７６と、ネーム自体のフィールド５７８と、Ｂ－ツリーポインタを収容するフィールド５８０とが含まれている。各ディレクトリ入口は、入口の長さを含んでいるワード５８２で始まる。これにより、各入口の終時におけるフレックススペースの可変量が与えられる。この可変量は、ファイルシステムの特別なバージョンに使用できるようにし、かつディレクトリブロックを極めて迅速に構切らせることを可能にする。

ディレクトリブロックの入口の数は、ネームの長さによって変化する。平均的なファイルネームの長さが１３文字であるときには、平均ディレクトリブロックはほぼ４０個の入口を保持するであろう。ディレクトリブロックの入口は、それらのネームフィールド（名前欄）の２進字句順序（binary lexical order）により分類される。最終入口は、ブロックの終了を印すダミーレコードである。

ディレクトリが大きくなり過ぎて１つのブロック内に記憶されなくなった場合には、Ｂ－ツリーとして編成される２Kbブロックを付加することによりサイズを大きくできる。特定のネームをサーチする場合には、ファイルシステムが一致を見出すか、目的とするネームよりも字句的に多いネームを見出すまで、ファイルシステムがディレクトリブロックを横行する。後者の場合、ファイルシステムが、入口からＢ－ツリーのポインタを抽出する。このポインタがどのサーチフィールドをも指示しない場合には、ファイルシステムは、ツリーにおける次のディレクトリブロックを次のポインタにより指示させ、サーチを続行する。

ブロック当り４０個の入口があるものと仮定すれば、ディレクトリブロックの２レベルツリーは1,640 個のディレクトリ入口を保持でき、３レベルツリーは65,640個の入口を保持することができる。換言すれば、最大限３つのディスクアクセスを備えた一般的な65,640個のファイルにおいて、特定のファイルを見出すことができる（又は、それが存在しないことを示すことができる）。ディスクアクセスの実際の数は、キャッシュコンテンツ（cache contents）及びディレクトリブロック

のB＋ツリーのファイルネームの位置に基づいて定められる。これは、最悪の場合4,000 個のセクタを読み取って、同数のファイルを収容しているディレクトリにファイルが存在するか否かを確定しなければならないFAT ファイルシステムに対して顕著な改善を与えるものである。

HPFSのB＋ツリーディレクトリ構成は、関数作業及び見出し作業に関するその効果を超える興味ある含意（implications）を有している。ディレクトリブロックを付加（又は除去）するか、或いはネームが収まるブロックから他のブロックに移動されてツリーのバランスを保つとき、ファイルの創出、リネーミング又は削除により、複雑な作業のカスケードを生じさせるかもしれない。実際、ファイル自体が成長することはないけれども、リネーム作業によってディスクスペースの不足をきたすであろう。この問題を回避するには、HPFSが、ディレクトリの緊急時に引き出すことができる空きブロックの小さなプールをリザーブし、このプールを指すポインタがスペアブロックに記憶されるように構成するのが好ましい。

ファイル属性は、ファイルの明白な記憶領域の外でオペレーティングシステムにより維持されるファイルについての情報である。

本発明のHPFSは、拡大属性（Extended Attributes 、「EAs」）を支持し、

ネーム＝バリュー

の形態をとる。但し、バリュー部分は、ゼロで終わる記号列（null-terminated string、「ASCIIZ」）でもよいし2進データでもよい点を除く。本発明の好ましい実施例においては、各ファイル又はディレクトリは、これに取り付けられたEAs の６４Kbの最大値をとることができる。但し、この制限は容易に変更することができる。

EAs の記憶方法は変えることができる。所与のファイル又はディレクトリと関連するEAs が充分に小さい場合には、これらのEAs はFnode に記憶されるであろう。また、EAs の全体のサイズが非常に大きいときには、これらのEAs はセクタ実行においてFnode の外に記憶され、割当てセクタのB＋ツリーが創出されて実行が記述される。単一のEAs が非常に大きい場合には、該EAs はFnode の外に押し出され、該EA自体のB＋ツリー内に押し込まれる。

本発明は、アプリケーションプログラムがファイルの拡大属性（extended atiributes）を定義することを可能にするOS/2カーネルのAPI 機能、すなわち、DOSQFileInfo及びDosSetFileInfoを改善することができる。また、本発明によれば、任意のパスネーム（パス名）と関連するEAs の読取り及び書込みに使用できる２つの新たな機能、すなわち、DOSQPathInfo及びDosSetPathInfoを得ることができる。アプリケーションプログラムは、特定のEA（これは、一致させるべきネームを供給する）のバリューを要求するか、ファイル又はディレクトリについての全てのEAs を一度に得ることができる、EAs の支持により、目的にかなったアプリケーションプログラムの使用が容易になる。ファイルを所有するアプリケーションのネームから、従属ファイルのネーム、アイコン及び実行コード（executable code）に至る殆ど全ての形式の情報をEAs に記憶させることができる。

HPFSは、多レベルでのディスクスループットの潜在的ボトルネックをアタックする。性能を向上させる為、HPFSは、進歩したデータ構成、連続セクタ割当て、インテリジェントキャッシング、読取りヘッド、及びデファード書込み（deferred writes）を使用している。最初に、HPFSは、そのデータ構成、すなわち、ファイルネーム、ディレクトリネーム、ファイル又はディレクトリに割り当てられたセクタのリストへの高速ランダムアクセスが行えるようにした複雑なデータ構成（Ｂ－ツリー及びＢ＋ツリー）、及び適当なサイズの空きスペースのチャンクを位置付けできるようにする簡単でコンパクトなデータ構成（ビットマップ）をタスクに一致させる。これらのデータ構成を操作するルーチンは、アセンブラ言語で記載するのが好ましい。

HPFSの主目的は、可能な限り、連続セクタをファイルに割り当てることである。ディスクの読取

り／書込みヘッドを或るトラックから他のトラックに移動させるのに要する時間は、可能性のある他の遅延よりも遙かに最大であり、このため、HPFSは、ファイルスペースを連続的に割り当てることにより、及びFnode及び該Fnodeが制御する単柄の近くの空きスペースビットマップのような制御構成を維持することにより、このようなヘッド運動を回避するか最小限にする。高度の連続的なファイルはまた、多くのセクタに対し一度に要求されるディスクドライバのリクエストを、ファイルシステムによって少なくすることを補助し、ディスクドライバが、ディスクコントローラの多セクタ移送能力を活用できるようにし、かつ、処理すべきディスクの完全な中断数を減減させることができる。

多数のファイルを同時に更新させるマルチタスキングオペレーティングシステムにおいてファイルが細分化されないように維持することは、従来技術には見られない特徴である。HPFSが用いている1つの方法（手順）は、新しく創出されたファイルを別々のバンドのディスクを横切って分散させ、できるならば、セクタが拡大されるときファイルに割り当てられるセクタがインターリーブされないようにすることである。他の方法は、ファイルを拡大しなければならない度毎に、連続スペースの4Kbをファイルに予め割り当てて、ファイルを閉じるときに全ての過剰のスペースを戻す方法である。

アプリケーションが、新しいファイルの最終的サイズを予め知ることができるならば、HPFSがファイルを創出するときに、最初のファイル割当てを特定化することにより、HPFSを補助することができるであろう。そうすれば、システムは全ての空きスペースビットマップをサーチして、ファイルを充分に保持できる連続セクタの実行を見出すことができるであろう。このことに失敗した場合には、システムは、ファイルのサイズの1/2である2ラウンドをサーチし、以下このことが繰り返される。

HPFSは、幾つかの異なる種類のキャッシングに頼って、HPFSが要求する物理ディスク転送の数を最小限にしている。HPFSは、FATファイルシステムが行ったようにして、セクタのキャッシングを行う。しかしながら、FATファイルシステムとは異なり、HPFSは非常に大きなキャッシュを効率良く管理し、セクタキャッシングを、パーハンドルベース(per-handle basis)で、ファイルが用いられる方法に調節するようになっている。また、HPFSは、パスネーム、ディレクトリ、トランスフォーミングディスクディレクトリ入口を、記憶表現(memory representation)におけるよりコンパクトで効率の良いものにキャッシングする。

性能を向上させるべくHPFSが用いられているもう1つの技術は、プログラムが必要とすると考えられるデータを予め読み取ることである。例えば、ファイルが開かれるとき、ファイルシステムが、Fnode及びファイルの内容の最初の幾つかのセクタを予めよみとりかつキャッシングする。ファイルが、該ファイル注の実行プログラム(executable program)又はヒストリー情報である場合には、Fnodeは、全ファイルを直ちに連続的に読み取ることにより、ファイルの開放作業が一般的に続けられていることを示す。ファイルシステムは、より多くのファイル内容物を用意しかつキャッシングする。プログラムが比較的少量の読取りリクエストを発行する場合には、ファイルシステムは2Kbのチャンクのファイルから絶えずデータを取り出し、過剰のデータをキャッシングする。このキャッシングにより、殆どの読取り作業が満足できるものになる。

本発明のHPFSは、OS/2のマルチタスキング能力に基づいた遅い書込み(lazy writes、デファード書込み又はライトビハインド(write behind)とも呼ばれている)を頼りにしているところが大きい。例えば、プログラムがディスク書込みを要求する場合には、データはキャッシュ内に置かれ、キャッシュバッファがダーティとしてフラグされる（すなわち、ディスクのデータの状態と一致しないことを示す）。ディスクがアイドル状態になるか、或いはキャッシュがダーティバッファで満

知される上、ファイルシステムは、デエモンプロセス（daemon process）からのキャプティブスレッド(captive thread)を用いてバッファをディスクに書き込み、最も古いデータでスタートする。キャプティブスレッド及びデエモンプロセスについては、Hastings、その他の著者によるテキストシリーズ「マイクロソフト社のOS/2プログラマーズリファレンス("Microsoft OS/2 Programmers Reference")」(1989 年、Microsoft Press 社刊)において説明されている。

一般に、遅い書込みは、プログラムがより高速で実行されることを意味している。なぜならば、一般に、プログラムの読取りリクエストは、書込みリクエストを待機して遅延することなく完了するからである。繰り返し読み取られるプログラムの場合は、小さなワーキングセットを読更して書き込むので、遅い書込みはまた、多くの不必要なすなわち冗長な物理ディスク書込みを回避することができる。遅い書込みはそれらの被る危険性を有しており、従って本発明は、DosOpen に対してOpenModeパラメータのライトスルーフラグ(write-through flag)を設定することにより、パーハンドルベース(per-handle basis)上で遅い書込みに打ち勝ち、DosBufReset 機能により、パーハンドルベース上のディスクにデータを委託(commit)することができる。OS/2の現行バージョンにおいても、DosOpen 及びDosBufReset の両機能を利用することができる。

遅い書込み（lazy write）の広範囲な使用により、HPFSが、あらゆる緊急事態の下での書込みエラーから優雅に回復できるようになる。例えば、書込みの失敗が知られるときまでに、アプリケーションは長時間を要する。なぜならば、アプリケーションは、データをディスク記憶装置内に安全に運び出したという錯覚の下で行われるからである。ディスクアダプタにより戻される「セクタが見つからないエラー("sector not found" error)」のようなエラーは、ハードウェアにより検出することができる。或いは、そのようなエラーは、データの書込み後、読取り検証（read-after-write verification)中、ハードウェアのディスクドライバにより検出される。

書込みエラーを取り扱う主要機構は、ホットフィックス（hotfix）と呼ばれる。エラーが検出されると、ファイルシステムは、リザーブされたホットフィックスプールから空きブロックを取り出し、該ブロックにデータを書込んで、ホットフィックスマップを更新する（ホットフィックスマップとは、単に、一連の対をなす二重ワードのことであり、二重ワードの各対は、そのホットフィックス交換の番号と関連する番号の悪いセクタを収容している）。次に、ホットフィックスマップのコピーがスペアブロックに書き込まれ、ディスク装置に問題があることをユーザに知らせる警告メッセージがディスプレイされる。

ファイルシステムがディスクドライバからのセクタ読取り又は書込みを要求する度毎に、ファンシステムは、ホットフィックスマップを走査して、悪いセクタの番号を実際のデータを保持している良いセクタに相当する番号に置き換える。

CHKDSKの1つのデューティはホットフィックスマップを空にすることである。ホットフィックスマップの各交換ブロックに対し、CHKDSKは、データを所有するファイルに対する好ましい位置にある新しいセクタを割り当て、データをホットフィックスブロックから新しく割り当てられたセクタに移動させ、かつ、ファイルの割当て情報（この情報には、再バランスしている割当てツリー及び他の精巧な作業が含まれている）を更新する。次いで、CHKDSKは、悪いブロックリストに悪いセクタを付加し、交換セクタを解放してホットフィックスプールに戻し、ホットフィックスマップからホットフィックス入口を削除し、かつ、更新したホットフィックスマップをスペアブロックに書き込む。

HPFSは、各HPFSボリュームのスペアブロックにダーティPSフラグを維持する。HPFSボリュームの全てのファイルが閉じられるとき、キャッシュ内の全てのダーティバッファが書き込まれるとき、或いは、ブートボリュームの場合にはシャットダ

ウンが選択されかつその作業を完了したときに、フラグがクリアされる。

OS/2のブートシーケンス中に、ファイルシステムが各HPFS上のダーティFSフラグを検査して、フラグが設定されている場合には、CHKDSKが実行されるまでブートボリュームには更にアクセスできないようにする。ブートボリュームにダーティFSフラグが設定されている場合には、システムは自動的にCHKDSKを実行する。

スーパーブロック又はルートディレクトリの損失というような真に重大な大事故の場合には、最も成功の可能性のあるデータ回復を与えることができるようにHPFSが設計されている。Fnode 、割当てセクタ及びディレクトリブロックを備えた始ど全ての形式の重要なファイル目的（オブジェクト）が、その親及び子の両方に二重にリンクされており、かつ、ユニークな３２ビットのサインを収容している。Fnode はまた、それらのファイル又はディレクトリのネームの最初の部分を収容している。従って、SHODS は、Fnode 、割当てセクタ及びディレクトリブロックに対してディスクを順回正しく走査し、Fnode 、割当てセクタ及びディレクトリブロックを用いてファイル及びディレクトリを再構成し、かつ最後に空きスペースのビットマップを再創出(regenerating)することにより、全ボリュームをリビルドすることができる。

上記のように、本発明は、ファイル及びディレクトリを論理的に順序付けするのに、B＋ツリー及びB－ツリー（２進ツリー）を用いている。２進ツリーは、データを物理的に順序付けすることなくして、ポインタを用いてデータ項目の集合を論理的に順序付けする技術である。

第５Ｐ図を参照すれば、簡単な２進ツリーにおける各ノードには、ツリーにおけるノードの論理的位置を決定するキー値を含む戴するデータと、並びにノードの左右のサブツリーを指すポインタとが設けられている。ツリーを開始するノードはルート（根）として知られており、ツリーの枝の端部に位置するノードは、ときどきリーフ（葉）と呼ばれている。

データの特定ピースを見出すには、２進ツリーがルートを横切るようにする。各ノードにおいて、所望のキーがノードのキーと比較される。両キーが一致しない場合には、所望のキーがノードのキーより小さいか大きいかに基づいて、ノードのサブツリーの１つのブランチ又は他のブランチが選択される。このプロセスは、一致が見出されるまで、又は第５Ｐ図に示すように空のサブツリーに出合うまで続けられる。

このような簡単な２進ツリーは、理解と実施が容易であるけれども、実用に際しては欠点を有している。キーが非ランダムな態様でツリーに首尾良く分散されなかったり付加されなかったりすると、ツリーが全く非対称的になり、ツリーの横断時間が広範囲に変化してしまう。

アクセス時間を均一にするため、多くのプログラマは、第５図に示すようなＢ－ツリーとして知られているバランス形ツリーを好む傾向にある。Ｂ－ツリーについての重要な点は、データが全てのノードに記憶され、１つ以上のデータ項目が１つのノードに記憶され、かつ、ツリーの全てのブランチが同じ長さをもっているということである。

Ｂ－ツリーの最悪の場合の挙動（behavior）は予測可能であり、簡単な２次ツリーの挙動より遙かに良好であるが、Ｂ－ツリーのメインテナンスはかなり複雑である。新しいデータ項目の付加、キーバリューの変更、又はデータ項目の削除により、ノードのスプリッティング（分割）又はマージング（併合）が生じ、これにより、ツリーには他の作業のカスケードが強制される。

第５Ｇ図に示すように、Ｂ＋ツリーは、２つのノード形式（内部ノードは他のノードを指すだけであり、外部ノードは実際のデータを有している）をもつＢ－ツリーの特殊化されたフォームである。

Ｂ－ツリーよりもＢ＋ツリーの優れている点は、Ｂ＋ツリーの内部ノードが、Ｂ－ツリーの中間レベルノードより非常に多くの決定バリューを保持でき、そのため、ツリーの外のファンが高速になりかつブランチの平均長さが短くなることである。これにより、必要データを見出すにはＢ＋ツリー

のブランチがその端部に終かなければならないという事実を維持でき、一方、B－ツリーにおいては、データは中間ノードにおいて発見され、或いは、ルート（根）においてさえも発見される。

本発明は、OS/2オペレーティングシステムを改善したものであり、多くのユーティリティとOS/2の現行バージョンにおいて利用できるサブルーチンを用いて実施することができる。本発明は、主としてOS/2オペレーティングシステムに使用することを意図しているが、本発明の原理は、実際のあらゆるコンピュータのオペレーティングシステムに適用できるものである。ここで説明する新しいユーティリティ及びサブルーチンを除き、他の全てのユーティリティ及びサブルーチンは現在利用されていて良く知られたものである。OS/2オペレーティングシステムの詳細な説明については、前述のOS/2プログラマ用参考書を参照されたい。本発明の改善されたOS/2オペレーティングシステムのボリュームマネージメント（volume management）は、OS/2の従来のバージョンにおいて行われているものと同じデューティ、すなわち、悪いボリュームがドライブにインサートされたときの検出、ボリュームが除去されたときの検出、ボリュームパラメータブロック(VPB)を介してドライブ内に置かれた新しいメディアに関する新しい情報の創出、適当なデバイスドライバとの通信、新しいインサートメディアにアクセスする必要のあるデバイス情報をシステムに与えること、バッファ及びFCOS 機構とのインターフェース、及び特定ボリュームへの変更をシステムに知らせること等に応答することができる。

OS/2の従来のバージョンにおいては、僅かに1つのファイルシステムがあったに過ぎない。本発明によれば、統一された環境内に多数のファイルシステムを設けることができる。ボリュームマネージャは、どのファイルシステムを特定のボリュームにアクセスさせるべきかを決定し、ファイルシステムドライバ（FSDs）が特定のボリュームについてのそれらの資源（resources）を管理（マネージ）できるようにする機構を提供し、かつボリュームの管理のために過去に設けられた全てのFSDsに対して同じサポートを提供する。本発明は、良く知られている既存のOS/2呼び出し(OS/2 calls) 並びに以下に説明する幾つかの新しい機能を有している。本発明の設置可能(Installable) なファイルシステムについての完全な説明は、マイクロフィッシュの形態で本願に添付されかつ参考として掲示する付録I（AppendixI）において述べられている。

本発明は、個々のボリュームについての正しいファイルシステムドライバの識別及びローディングが容易に行えるマウントプロセス及びアンマウントプロセスを用いることを意図している。

マウントプロセスは、幾つかの異なる事象が生じたとき、すなわち、

1．ボリュームへの最初のアクセスがあったとき、

2．ドライブのボリュームが不確実になる（このことは、通常、ユーザが新しいメディアをドライブに入れることを意味する）全てのとき、

3．ドライブ内にないボリュームへのアクセスが要求される全てのとき、

に開始される。

マウントプロセスへの入力は、ドライブパラメータブロック（DPB）（このドライブパラメータブロックは、デバイスドライバにI/O を行うこと、及びドライブ内にあると現在考えられているボリュームのVPB にハンドルを記憶させることに使用される）を指すポインタである。マウント作業によりこれが更新される。ローカルVPB がスタック上に割り当てられ、DPB ポインタと共に初期化される。

第6図に示すように、項目602で示すようにメディアの論理的セクタ0を読み取ることにより、マウントプロセス600が開始される。デバイスドライバからの全てのエラーは無視する。なぜならば、異なる形式のメディア（すなわち、光学ディスク又はCD-ROM）がトラック0を読取り不能にできるからである。論理的セクタ0を読み取る前に、テンポラリマウントバッファが0に初期化さ

れる。ボリュームのラベルテキストフィールドが「UNLABELED」に初期化される。セクタ0がチェックされ、特定バリュー（41）に対してサインバイト（signature byte）を比較することにより、フォーマットが認識されているか否かを決定する。フォーマットが認識されていない場合には、VPBに近い情報がスタック上に丸裸される（すなわち、32ビットボリューム連続番号（32 Bit Volume Serial Number））。

次に、項目604により、BUILDBPB呼び出し（BUILDBPB call）が、DPBにおいて特定化されたデバイスドライバに発行される。BUILDBPBは、デバイスドライバによりエクスポート（移出）される手順である。このBUILDBPB手順については、付録Iにおいて詳細に説明されている。BUILDBPBは、装置の物理パラメータ（バイトパーセクタ（byte per sector）、セクタパートラック（sector per track）等）を学習させるべく呼び出される。デバイスドライバは、デバイスドライバは、ボリュームの物理パラメータを決定するのに用いることができる情報を収容しているバッファを指すポインタに導かれる。幾どのドライバにとっては、これはセクタ0であり、非常に古い幾つかのドライバにとっては、FATの最初のセクタである。装置が、セクタ0から読み取られるデータを解釈できない場合（例えば、この場合のフロッピがFATではなく。従ってFAT IDバイトが意味をもたない場合）には、装置は最小のBPBを戻し、カーネル及びFSDsが必要なI/Oを行って、ボリュームを完全に識別できるようにする。

前に創出されたBPBからの適合フィールド（relevant field）は、スタックのローカルVPB（すなわち、Sectors/track、NumberofHeads、Total Sectors、Secior Size）にコピーされる。新しいVPBが割り当てられ、ローカルVPBからの情報がそれにコピーされる。次に、本発明によれば、ループ606に入り、項目608で示すように、新しく創出されたVPB、論理的セクタ0を指すポインタ、及びVPBファイルシステムの独立及び従属領域（independent and dependent areas）を指すポインタを備えたFS_MOUNT（フラグ＝0）の入口点を呼び出すことにより、各FSDをポール（poll）する。FSDはFSH_DoVolIOを呼び出し、ボリュームから他のセクタを読み取る（それ自体のバッファを割り当てなくてはならない）。FSDが、ERROR_UNCERTAIN_MEDIAに戻る場合には、エラーが戻され、プロセスは、決定（decision）610により示すように再スタートされる。FSDがブートセクタをサポートする場合には、FSDは、ブートセクタのファイルシステムのネームフィールドをチェックして、これがネームフィールドを認識しているか否かを決定する。FSDがブートセクタをサポートしない場合には、FSDがボリュームを認識しているか否かを決定すべく、装置へのI/Oが行われる。FSDがひとたびボリュームを認識しているならば、項目612で示すように、VPBファイルシステムの独立及び従属領域における適合フィールドを更新する。VPBファイルシステムの独立及び従属領域については、第7図に関連して更に詳細に説明する。この時点においては、FSDはFS Helper(FSH)機能を発行して、新しいボリュームが、本発明が管理する他の任意のボリュームと同じであるか否かを決定する。このFS Helperは、ファイルシステムの独立及び従属領域にポインタを戻す。次いでFSDは、項目614で示すように、新しく創出されたVPBから古いVPBへと情報をコピーする。新しく創出されたVPBは、MOUNTの呼び出しを行った後に破壊される。次に、FSDは、あらゆるバッファを無効にするような古いVPBに対してあらゆるクリナップ作業（cleanup work）を行う。これは、ドライブからボリュームが除去されていることがあるからである。

本発明によれば、ひとたびFSDがボリュームを認識すると、リスト内に一致するものが見出される場合には新しいVPBが除去される。リスト内に一致するものが見出されない場合には、VPBは、マウントされたFSDsのリストにリンクされる。FSDsが認識されない場合には、決定614及び項目616に示すように、VPBが空にされかつFATファイルシステムがマウントされる。

新しいボリュームがドライブにインサートされかつ古いボリュームに対してカーネルがもはや関与しない場合には、本発明では、FSD にFS_MOUNT（フラグ＝2）が発行され、これにより、このボリュームに割り当てられた資源の割当てを解除するようになっている。

本発明により、新しくインサートされたボリュームが、ドライブの最終ボリュームとは異なるものであることが検出された場合には、FSD に対しFS_MOUNT（フラグ＝1）の呼出しが発行され、これにより、除去されたボリュームに関するバッファ無効のようなあらゆるクリナップ形式の作業を行うことができる。ボリュームに対してもはやカーネルが関与しない場合には、FS_MOUNT（フラグ＝2、UNMOUNT）が続いて行われる。新しくインサートされたボリュームが、ドライブにおいて最終的に見出されたボリュームと同じものである場合には、この呼出しは発行されない。

本発明は、FSD により要求される機能に対する既存のカーネル資源を利用するのに、効率の良い機構を用いることを意図するものである。より詳しくは、FSD がカーネル内に存在する機能を要求する場合には、FSD は、ファイルシステムヘルパ（FSH）を呼び出す（invoke）ファイルシステムヘルパ呼出し（call）を発行する。呼び出されたFSH は、次に、要求された情報を戻す。以下に、ファイルシステムヘルパについて簡単に説明する。以下に述べる要約においては幾つかの重要なファイルシステムヘルパがリストアップされているけれども、必要に応じて付加的なファイルシステムヘルパを設けることができる。ファイルシステムヘルパは、Appendix I において詳細に説明されている。

ファイルシステムヘルパ（File System Helpers）

FSH_GETVOLPARM：多くのFS呼出し時に、VPB へのハンドルがFSD に渡され、FSD が、VPB のファイルシステムの独立及び従属領域にアクセスすることがしばしば必要になる。このヘルパは、そのようなサービスを与えるものである。

FSH_DOVOLIO：FSD が、特定のボリュームに対してI/O を行う必要があるときには、FSD は、このヘルパを使用して、要求されたボリュームが実際にドライブ内にあることを保証し、適当なデバイスドライバを呼び出し、かつハードエラーを取り扱う。このヘルパは、FSD 内で常時用いることができる。FS_MOUNT 呼出しの範囲内で呼び出されるとき、FSD はドライブのボリュームに適用される。しかしながら、FSD がFS_MOUNT 呼出しに戻るまでは、ボリュームの認識が完了していないので、ERROR_UNCERTAIN_MEDIA が戻される場合には、FSD に注意しなければならない。このことは、ドライブにおけるメディアの識別を試みる間に、メディアが不確実になっていることを示すものである。また、このことにより、FSD が認識を試みていたボリュームが除去されたことを示すようにすることもできる。この場合には、FSD は、FS_MOUNT 呼出しに渡かれたhvPBに取り付けられた（attached）全ての資源を解放し、ERROR_UNCERTAIN_MEDIA がFS_MOUNT 呼出しに戻される。これにより、ボリュームトラッキング論理が、マウントプロセスを再スタートするように命令される。

FSH_DUPLICATEVPB：FS_MOUNT 呼出しの間、入力VPB は、管理されている他の1つのボリュームと同じボリュームにすることができる。新しいボリュームに関する更新情報を創出しかつ古い複製VPB（older duplicate VPB）への情報をコピーすることは、FSD の責任である。このヘルパは、古い複製VPB が存在しているか否かを決定し、存在している場合には、古い複製VPB のファイルシステムの独立及び従属領域を指すポインタを戻し、これらの領域がFSD によって更新されるようにする。次いで、FSD は、ボリュームが除去されているので、古いボリュームについてあらゆるクリナップ作業を行う。

上記のように、本発明は、可能な限り、予め存在しているOS/2資源を使用することを狙ったものである。以下のリストは、本発明の作業中に呼び出される機能の階層（hierarchy）を要約したものである。

```
1       DoVolIO
1.1         WhatVolume
```

1.1.1	ProbeChange
1.1.2	ResetMedia
1.1.3	GenhVPB
1.1.3.1	LockVBuf
1.1.3.2	HeadBoot
1.1.3.3	BuildBPB
1.1.3.4	FSMountVolume
1.1.3.4.1	Bmp_Get
1.1.3.4.2	VPBCopy
1.1.3.4.3	VPBLink
1.1.3.4.4	VPBFind
1.1.3.4.5	VPBFree
1.1.3.5	SetVPB
1.1.3.6	FindVID
1.1.3.7	DiskIO
1.1.3.8	CRC
1.1.3.9	VPBFIND
1.1.3.10	Bmp_Get
1.1.3.11	VPBCopy
1.1.3.12	VBPLink
1.1.3.13	UnlockVBuf
1.1.3.14	Buflnavalidate (Redetermine Media)
1.1.3.15	FlushBuf (Redetermine Media)
1.1.4	IncVPBRef
1.1.5	DecVPBRef
1.1.5.1	VPBFree
1.1.6	ResetCurrency
1.1.6.1	NextCDS
1.1.6.2	PointComp
1.1.6.3	Buflnvalidate

テーブル　1.

本発明によれば、メディアが不確実であるか否か又はメディアが最初にアクセスされたか否かが呼び出される。本発明のボリュームマネージメント機能（ボリューム管理機能）はライン1．により表される。最初のプロセスは、ライン1.1 で示すように、どのボリュームがシステムに表されたかを決定することである。ライン1.1.1 のProbeChangeは、デバイスドライバにアクセスすべく呼び出され、メディアの変化をデバイスドライバが検出したか否かを決定する。メディアの変化が検出されたときは、ライン1.1.2 においてResetMedia が呼び出され、デバイスドライバがメディアにI/O できることを知らせる。次に、ライン1.1.3 においてGenhVPB が呼び出され、ボリュームパラメータブロックが創出される。このプロセスは、LockVBufが呼び出されてオペレーティングシステムのカーネル内のバッファをクリアしかつ直列化(serialize) するライン1.1.3.1 と共に開始する。ライン1.1.3.2 においては、メディアブートセクタのデータが、オペレーティングシステムのバッファに読み取られる。システムはライン1.1.3.3 に続き、該ライン1.1.3.3 においては、BuildBPBが呼び出されて(invoked) 、ディスクドライバを呼び出し(call) 、ブートパラメータブロックを作る。次に、FS_MOUNTがライン1.1.3.4 に呼び出される。FS__Mount における最初のステップは、ライン1.1.3.4.1 のBmp_Get を呼び出す。ライン1.1.3.4.1 は、BPB のバッファを設定すべく呼び出されるカーネルにおけるメモリマネージメントユーティリティである。ライン1.1.3.4 においては、FSMountVolume が呼び出されると、FSMountVolumeは、FSDsのリストを介して反復し、サクセスに戻るかリストの終部（エンド）に到達するまで、各FSD のFS__Mount 手順を呼び出す。ライン1.1.3.4.2 においてFSD がサクセスに戻ると、VPBCopyが呼び出されて、BPB のコピーに対するテンポラリバッファを創出する。次に、ライン1.1.3.4.3 におけるVPBLink が呼び出され、VPB をチェーンにリンクさせ、該チェーンにおける次のVPB を指すべくBPB を設定し、現在のVPB をリストの開始に初期化する。VPBFind がライン1.1.3.4.4 に呼び出されてVPBsのチェーンを試験し、プロセス中のVPB と同じボリューム識別子を所有しているVPB を見出す。複製VPB の識別子が見出された場合には、ライン1.1.3.4.5 にVPBFree が呼び出され、複製VPB がVPBsのリスト内に見出された場合には、試験を受けてVPB がBPB から自由に

なる。FSMountVolumeが完了すると、VPBに適当なフィールドを設定するライン1.1.3.5にSetVPBが呼び出される。ライン1.1.3.6において、FindVIDが呼び出され、ボリューム識別子が見出される。メディアのセクタ0にブートブロックが見出されない場合には、ライン1.1.3.7にDisk10が呼び出され、ボリュームのBPBが位置付けされる。FSDのFS_Mountルーチンがサクセスに戻らない場合には、(残留) FATファイルシステムのFS_Mount手順と論理的に等価のインラインコードが呼び出される。ライン1.1.3.8においては、CRCが呼び出されて、古いFATボリュームの最初のディレクトリが検査合計(checksum)され、それらのブートセクタの連続番号をもたないボリュームについてのユニークなボリューム連続番号が創出される。次に、ライン1.1.3.9～ライン1.1.3.13にリストアップされた機能が呼び出されて、新しいボリューム識別子が創出されかつボリューム識別子バッファが空にされる。ライン1.1.2.14においてはBufInvalidateが呼び出されて、プロセス開始以来メディアが変化している場合にはバッファ内の全てのデータが無効にされる。その場合には、ライン1.1.3.15にFlushRefが呼び出され、新しいメディアに対してバッファをフラッシュさせる。

ボリュームについて前から存在しているVPBが見出されない場合には、ライン1.1.4のIncVPBRefが呼び出されて、現在のVPBの基準カウンタが増大(increment)される。この基準カウンタは、ここで問題にしているボリュームが、オペレーティングシステムのカーネルに対して依然として開放しているか否かを記録するのに使用される。ライン1.1.5においては、DecVPBRefが呼び出され、前のVPBの基準カウンタが減少(decrement)される。基準カウンタがゼロに減少された場合には、VBPFreeがライン1.1.5.1に呼び出され、VPBが空にされる。ライン1.1.6にはResetCurrencyが呼び出され、現在のディレクトリ構成における位置データが無効であるとしてマークする。NextCDS(ライン1.1.6.1)及びPointComp(ライン1.1.6.2)は、現在のディレクトリ構成(CDSs)を列挙するのに用いられる内部ルーチンである。ライン1.1.6.3においてBufInvalidateが呼び出され。ファイルシステムのバッファプールから、(現在は無属化している)VPB基準が除去される。

上記のように、VPBは、コンピュータシステムに使用されている特定のボリュームについての情報を記憶するシステムにより用いられる。ボリュームは、ブロック装置におけるメディアとして構成され、メディアに関する情報は、このボリュームを他の全てのボリュームから区別する。

VPBsは、BMPとしてセグメントに維持される。従って、システムは記録が使用されているトラックのみを必要とし、かつ空のリストが管理される。

新しいボリュームに出合う度毎に、すなわち、ボリュームのVPB作製(VPB built)がシステムに既存のいかなるVPBsとも一致しない度毎に、新しい入口(new entry)が、BMP管理されたセグメント内で割当てられ、メディアからの等価データで充填される。システムがVPBで完了する度毎に、すなわち、システムのRefCountがゼロになる度毎に、BMP管理されたセグメントの入口が空になり、BMPは、この空になった記憶機構(storage)を再使用のためにトラックする。テーブル1の機能により用いられた構成を以下に説明する。

VPBは、次の3つの部分に分割される。

1．カーネルのプライベート部分。この部分は、情報をカーネルに維持するのに使用され、VPB(例えば、基準カウント)の管理を必要とする。これは、カーネルにとってのプライベートなものであり、FSDsは決してこのプライベート部分にアクセスしないしかつこれを変更することがないことを意味している。

2．ファイルシステムの独立部分。この部分は、全てのファイルシステムにより使用されかつ特定のあらゆるファイルシステムから独立している。この部分は、或るファイルシステム(file system、「FS」)の要求に応じて、装置可能なファイルシステムに導かれる。

3．VPBを用いているファイルシステムに特有の部分。この部分は、必要に応じてファイルシス

テムを使用できる「作業領域」として設定される。この部分は、成るFSの要求に応じてIFS に導かれる。VPB のレイアウトは第7図に示されている。

次の構成は、ファイルシステムのVPB とは独立した部分を明らかにするものである。この構成は、ファイルシステムの形式の如何に係わりなく、あらゆるファイルシステムにより使用できるものである。

```
vpbfsi          STRUC
vpi_ID          DD    ? ;ファイルの32ビットユニークID
vpi_pBPB        DB    ? ;ドライブボリューム内蔵
vpi_cbSector    DW    ? ;バイトの物理セクタのサイズ
vpi_totsec      DD    ? ;メディアのセクタの全数
vpi_trksec      DW    ? ;メディアのトラック当りのセクタ
vpi_nhead       DW    ? ;装置のヘッドの数
vpi_tmxt        DB    VPBTEXTLEN DUP(?) ;ユーザ用のプリントできる可能なID
vpbfsi  ENDS
```

下記の構成は、VPB のファイルシステムの従属部分を定めるものである。この構成は、適合すると考えられるファイルシステムにより使用される。

```
vpbfsd  STRUC
vpd_work        DB  VOBWORKAREASIZE DUP (?)
vpbfsd  ENDS
```

下記の構成は、ボリュームパラメータブロック(VPB)の構成を定めるものである。

```
vpb             STRUC
```

全てのファイルシステムについてカーネルにより使用されるフィールド(欄)

```
vpb_flink       DW    ? ;前方リンクのハンドル
vpb_blink       DW    ? ;後方リンクのハンドル
vpb_fSsector    DD    ? ;IOのセクタ数
vpb_ref_count DW    ? ;VPB を指す目的のカウント
vpb_search_count DW   ? ;VPB を指すサーチのカウント
vpb_first_access DB   ? ;これは、メディアを強制すべく-1 に初期化される
vpb_signature DW    ? ;VPB の有効性を明記するサイン
vpb_flags       DB    ? ;フラグ
vpb_FSC         DD    ? ;ファイルシステム制御ブロック(FSC) を指すポインタ
```

下記のフィールド(欄)は、ファイルシステム従属作業に使用される。

```
vpb_fsd         DB    SIZE vpbfsd DUP (?)
```

下記のフィールドは、ファイルシステム独立作業に使用される。

```
vpb_fsi         DB    SIZE vpbfsi DUP (?)
vpb             ENDS
```

下記の構成は、FSH_GETVOLPARM(これは、VPB ハンドルからVPB データを得るのに使用される)により使用される。

```
ENTRY   push    word hVPB               (1 word)
        push    dword ptr to file
                system ind.             (2 word)
        push    dword ptr to file
                system dep.             (2 word)
        call    FSHGETVOLPARM
EXIT  (ax) = return code  (リターンコード)
          0-success
```

下記の構成は、FSH_DOVOLIO (これは、ボリュームベース形セクタ配向転送(volume-based sector-oriented transfers)に使用される) により使用される。

```
ENTRY   push    word Operation          (1 word)
        push    word hVPB               (1 word)
        push    dword ptr to user
                transfer area           (2 word)
        push    dword ptr to sector
                count                   (2 word)
        push    dword starting sector
```

```
                number                  (2 word)
        call  FSHDOVOLIO
EXIT  (ax) = return code
              0-success
```

下記の構成は、FSH_DUPLICATEVPB（これは、複製（古い）VPBへのVPB データを得るのに使用される）により使用される。

```
ENTRY   push   word hVPB                (1 word)
        push   dword ptr to file
                system ind.             (2 word)
        push   dword ptr to file
                system dep.             (2 word)
        call   FSHGETVOLPARM
EXIT  (ax) = return code
              0-success
```

HedeterelaeMediaは、下記に示すような特殊な組の入口パラメータ(entry parameters)を有している。

```
ENTRY (DS;SI) dpb への点(point)
EXIT          Carry clear = >
                (DS;SI).hVPBは、「正しい」ボリュームで充填される。
              Carry Set = >
              (AX) = I/O packet status;作業は失敗した。
USES          AX, BX, DX, CI, BS, Flags
```

下記の呼び出しは、ボリューム管理の内部コンポーネントインターフェースに使用される。

GeahVPB は、特定のドライブの内部VPB の決定に使用される。戻された全てのエラーは、ユーザに送られる。

```
Inputs:  ds:si  関心をもつDPB への点。これ及びこの中の全てのボリュームはロックされる。
Outputs; Carry clear = >axは、ドライブを行うVPB へのハンドルである。
         Carry set = >作業（オペレーション）がエラーを創出した。
         zero clear = >作業は失敗した。
```

```
         zero set = >発生した不確実メディアを入れ子にした(nested)。
```

全てのレジスタは、変更することができる。

BuildBPB は、古いディスク（すなわち、認識されたブートセクタを有していないディスク）用の正しいBPB を創出すべく呼び出される。新しいディスクは、ブートセクタ内に、KNOWN(既知）で正しいすなわち正当なBPB(VALID BPB)を有している。デバイスドライバへのバッファは、BuildBPB 呼び出しの一部である。

```
Inputs:  ds;si  関心をもつDPB への点。
                pVPBBuf は、ロックされる。
Outputs; carry clear = >
         ds:si BPBを指す。
         carry set = >
         (AX) = 装置からの状態ワード(status word)
         zero set = >不確実メディアを入れ子にした。
         zero reset = >作業は失敗した。
```

全てのレジスタは、BPを除く全てを変更した。

FSMountVolume は、IFS ドライバが、関心をもつボリュームを認識しているか否かを決定すべくチェックする。

FSMountVolume は、各FSドライバのFS___Mount入口点を呼び出すFSD チェーンを通してループし、IFS が、関心をもつボリュームを認識しているか否かを決定する。このループは、最初のIFS がボリュームを認識するとき、又はシステム内に設置されたFSドライバの番号のループカウンタが 0に減少するときに、終端する。

```
Inputs:  ds;bx pVPBBufブートセクタへの点。
         di   スタックのLocalVPBのオフセット
Outputs; di = IFS がボリュームを認識した場合のFSC へのオフセット
         di = -1 。いかなるIFS ドライバもボリュームを認識しなかった場合
         ax = vpb ハンドル
変更されたレジスタ; ax, bp, bx, di, es, si, ds
```

VPBFree は、リンクリストからVPB を除去して、セグメントからそのブロックを空にする。

ENTRY　(BP) ＝VPB へのハンドル

EXIT　リンクされておらずかつ空にされたVPB

USES　bx, bp, cx, di, ds, es

VPBLink は、リストの開始時に新しいVPB をインサートし、新しいVPB 及び古い最初のVPB の前方及び後方のリンクフィールドを調節する。

ENTRY　ES:DI ＝新しいVPB

EXIT　リストにリンクされたVPB

USES　DS, SI

VPBFind は、内部リストを走査して、入力VPB と同じボリュームIDをもつVPB を探す。

ENTRY　DS:SI ＝入力VPB のボリュームIDを指すポインタ

EXIT　AX＝hVPB、見出された場合
AX＝0 、見出されない場合

USES　AX, BX, CX, DI, DS, ES

VPBCopy は、ローカル領域からBMP 管理領域にVPB をコピーし、かつ正当であるとしてVPB をスタンプする。

ENTRY　SI ＝スタックのLocalVPBのオフセット
ES:DI ＝＞新しいVPB

EXIT　なし

USES　AX, CX, DS, SI

悪いボリュームがマウントされたときにこれを検出してオペレータに正しい処置をとるように知らせること、すなわちボリュームマネージメント（ボリューム管理）は、オペレーティングシステムのカーネル及び適当なデバイスドライバを介して直接行われる。本発明の原理によれば、各ファイルシステムドライバ（FSD）は、ボリュームラベルと、ファイルシステムに使用される各ボリュームについての３２ビットのボリューム連続番号とを創出するようになっている。これらは、ボリュームがフォーマット化されるときに、論理的セクタゼロのリザーブされた位置に記憶させるのが好ましい。この情報を記憶するのに、特別のフォーマットが必要になることはない。オペレーティングシステムのカーネルは、FSD を呼び出して、これを含むことがある作業を実行する。FSD は、ボリュームラベル又は連続番号が変更された場合には、いつでもボリュームパラメータブロック（VPB）を更新する。

FSD が I/O リクエストをFSヘルパルーチンに導くと、デバイスドライバは、３２ビットのボリューム連続番号及びボリュームラベルを（VPB を介して）導く。ボリュームに関してI/O が実行されるとき、オペレーティングシステムのカーネルは、リクエストされたボリュームの連続番号を、装置を維持している現在のボリュームの連続番号と比較する。これは、ドライブにマウントされたボリュームのドライバパラメータブロック（BPB）のVPB をチェックすることにより行われるインストレージ試験（in-storage test）であり、いかなるI/O も必要とされない。比較の結果、等しくない場合には、オペレーティングシステムのカーネルは、クリチカルエラーハンドラに信号を発生して、特定の連続番号及びラベルをもつボリュームをユーザがインサートすることを促進させる。

メディアの変化が検出されると、アプリケーションプログラムのインターフェース（API）の機能呼び出しの代わりに、ドライブがアクセスし、本発明によれば、そのボリュームに対するマネージングI/O に応答できるファイルシステムドライバ（FSD）が決定される。次に、本発明は、ボリュームパラメータブロック（VPB）を創当て、設置されたFSDsをポーリング（poll）する。FSD は、これがメディアを認識していることを表示する。FSDsは上記のようにしてポーリングされる。

PAT のFSD は、FSDsのリストの最後にあり、他のFSD の認識が生じない場合には、全てのメディアを認識することにより、省略時FSD（default FSD）として作用する。

本発明の原理によれば、ファイルシステムドライバには、次の２つのクラス、すなわち、

１、ローカルすなわち遠隔（仮想ディスク）装置に対してI/O を行うブロックデバイスドライバを用いているFSD（これは、ローカルファイルシステムと呼ばれている）と、

2、ブロックデバイスドライバを用いることなくして遠隔システムにアクセスするFSD（これは、遠隔ファイルシステムと呼ばれている）とがある。

ドライブレター（ドライブ文字）と遠隔ファイルシステムとの間の連結はプログラムされたインターフェースを介して行われる。システムのネームスペース（例えば、ドライブ）の目的とFSD との間の結合を生じさせるには、DosFSAttach システムの呼出しが用いられる。

疑似文字装置（pseudo-character device）と遠隔ファイルシステムとの間の連結も、DosFsAttach インターフェースを介して行われる。DosFsAttach インターフェースは、DosFsAttach 呼出し及び DosQFsAttach呼出しで構成されており、これらは、Appendix Iにおいて詳細に説明されている。

ローカルボリュームが最初に参照されるとき、本発明では、FSD チェーンの各ローカルFSD を連続的に尋ねて(ask)、各FSD のFS_MOUNT 入口点への呼出しを介してメディアを受け入れるようになっている。どのFSD もメディアを受け入れない場合には、メディアは、省略時ファイルシステムに割り当てられる。FORMATでは認識されないメディアにアクセスするため行われる他の全ての試みがなされて、「無効（正しくない）メディアフォーマット（INVALID MEDIA FORMAT）」のエラーメッセージが出される。

ひとたびボリュームが認識されると、ドライブと、FSD と、ボリュームの連続番号と、ボリュームのラベルとの間の関係が記憶される。ボリュームの連続番号及びラベルは、ボリュームのパラメータブロック(VPB) に記憶される。VPB は、開放ファイル(I/Oに基づくファイルハンドル)、サーチ及びバッファリファレンス(buffer references)についてのオペレーティングシステムにより維持される。

除去されたボリュームに対する連続リクエストは、FS_MOUNTを呼び出すことによりボリュームについての設置されたFSDsのポーリングを要する。認識しているFSD により戻されたVPB 及び既存のVPB のボリューム連続番号及びボリュームラベルが比較される。試験が成功した場合には、ボリュームにFSD がアクセスされる。これに対し、試験が失敗した場合には、オペレーティングシステムがクリチカルエラーハンドラに信号を伝達し、ユーザがボリュームを矯正することを促進させる。

メディアとVPB との間の連結は、ボリュームの全ての開放ファイルが閉じられるまでセーブされ、サーチリファレンス及びキャッシュバッファリファレンスが除去される。ボリュームの変化によってのみ、次のアクセス時にメディアの再決定がなされる。

ブート可能で論理的に区分（partition）されたメディアに関するオペレーティングシステムの区分へのアクセスは、OS/2オペレーティングシステムに利用できる機能セット（機能組）のようなフルオペレーティングシステムの機能セットを介して行われる。ディスクの区分化の設計(disk partitioning design) についての詳細な説明が、前述のOS/2プログラマ用参考書においてなされている。

本発明によれば、ネットワークを介してオペレーティングシステムと通信する遠隔装置を識別するためのDosQFsAttach機能が提供される。DosQFsAttach の目的は、取り付け(attach)られた遠隔ファイルシステム、ローカルファイルシステム、文字装置(character device)、又は、ローカルFSD 又は遠隔FSD に取り付けられた疑似装置ネームに関する情報を尋ねることである。

DosQFsAttach を呼び出すシーケンスは次の通りである。

```
EXTERN DosQFsAttach: FAR
PUSH ASCIIZ DeviceName   ;デバイスネーム又は"d:"
PUSH WORD   Ordinal      ;ネームリストの入口の序数(ordinal)
PUSH WORD   FSAinfoLevel ;要求される、取り付けられたFSD データの形式
PUSH OTHER  DataBuffer   ;戻されたデータバッファ
```

```
PUSH WORD    DataBufferLen;バッファ長さ
PUSH DWORD   0             ;リザーブド(0でなくてはならない)
CALL DosQFsAttach
```

ここで、DeviceNameは、コロン(:)を付した駆動文字(drive letter)を指すか、或いは文字又は疑似文字装置ネームを指し、FSAInfoLevelの幾つかのバリューは無視する。DeviceNameが文字又は疑似文字であるときには、DeviceNameは、コロンを付した駆動文字の形態を有するASCIZ ストリングである。DeviceNameが、文字又は疑似文字の装置ネームであるときには、そのフォーマットは、サブディレクトリ呼出しのファイルネームのフォーマットにおけるASCIIZストリングのフォーマットであり＼DEV＼のように示すのが好ましい。

序数(ordinal)は、文字装置、疑似文字装置又はドライブの組のリストのインデックスである。序数は常に1からスタートする。リストの1つの項目の序数位置は重要でない。序数は、リスト全体を厳格にステップするのに使用される。序数から項目へのマッピングは揮発性であり、DosQFAttachに対する或る呼出しから次の呼出しまで変化することができる。

FSAInfoLevelは、要求される情報のレベルであり、DataBufferのデータがどの項目に関するものであるかを決定する。

レベル0×0001は、DeviceNameにより名付けられた特定のドライブ又は装置ネームのデータを戻す。序数のフィールド(欄)は無視する。

レベル0×0002は、序数により選択された文字又は疑似文字装置のリストの入口のデータを戻す。DeviceNameのフィールドは無視する。

レベル0×0003は、序数によって選択されたドライブのリストの入口のデータを戻す。DeviceName のフィールドは無視する。

DataBufferは戻り情報バッファ(return information buffer)であり、次のフォーマット内にある。

```
struct {
                unsigned short iType;
                unsigned short cbName ;
                unsigned char  szName  [] ;
                unsigned short cbFSDName;
                unsigned char  szFSDName [] ;
                unsigned short cbFSAData;
                unsigned char  rgFSAData [] ;
} ;
```

iType	項目の形式
1＝常駐文字装置(Resident character device)	
2＝疑似文字装置	
3＝ローカルドライブ	
4＝FSD に取り付けられた遠隔ドライブ	
cbName	項目ネームの長さ、ゼロ(null)は数えない
szName	項目ネーム、ASCIIZストリング
cbFSDName	FSD ネームの長さ。ゼロは数えない
szFSDName	ASCIIZストリングに取り付けられたFSD 項目のネーム
cbFSAData	FSD により戻されたFSD アタッチデータの長さ
rgFSAData	FSD により戻されたFSD アタッチデータ

szFSDName は、FSD により移出(エクスポート)されたFSD ネームであり、このネームは、必ずしもブートセクタのFSD ネームと同じネームである必要はない。

ローカル文字装置(iType＝1)については、cbFSDName ＝0であり、szFSDName は、ゼロで終わるバイトのみを含んでいて、cbFSAData ＝0である。

ローカルドライブ(iType ＝3)については、szFSDName は、呼出しの時点でドライブに取り付けられたFSD のネームを含んでいる。この情報はダイナミックに変化する。ドライブがオペレーティングシステムのカーネルの常駐ファイルシステムに取り付けられている場合には、szFSDName に"FAT"又は"UNKNOWN"が含まれるであろう。常駐ファイルは、マウントを扱むFSDs以外の任意のディスクに取り付けられるので、認識可能なファイルシステムを含んでいないディスクを設けること

ができるが、常駐ファイルシステムに取り付けることもできる。この場合、差異を検出することができ、この情報は、破壊されないデータのプログラムが、適正に認識されなかったディスクに存在することを助ける。

DataBufferLen は、戻りバッファのバイト長さである。戻り時においては、この長さは、FSD によりDataBufferに戻されたデータの長さである。

Returns: IF ERROR (AX not = 0)

AX = Error Code:

ERROR_INVALID_DRIVE - 特定のドライブは無効である（正しくない）。

ERROR_BUFFER_OVERFLOW - 特定のバッファは、戻されたデータにとって非常に短い。

ERROR_NO_MORE_ITEMS - 特定の序数は、リストにない項目に関するものである。

ERROR_INVALID_LEVEL - 正しくない情報レベル（Invalid Info level）

全てのブロック装置及び全ての文字及び疑似文字装置についての情報は、DosQFsAttach により戻される。この呼出しにより戻される情報は、揮発性が強い。

戻された情報は、これらが戻されるときまでに既に変化されていることを呼出しプログラムが気付くことができるようにするのが好ましい。カーネルの常駐ファイルシステムに取り付けられたディスクに戻された情報は、ディスクがファイルシステムを備えたものであることをカーネルが明確に認識しているか否かの決定、又はカーネルがそのファイルシステムをカーネルに取り付けたか否か（他のFSDsはディスクを取り付けていないからである）の決定を行うのに使用することができる。

全てのFSDsへのエラー信号についてのエラーコードの組は、0×E800～0×EEFFである。必要に応じて他のエラーを付加できるけれども、次のエラーが定められている。

ERROR_VOLUME_NOT_MOUNTED = 0×E800-FSDは、ボリュームを認識しなかった。

各FSD により定められるエラーコードの組は、0×EF00～0×EFFFである。

ディスクメディア及びファイルシステムのレイアウトは、次の構成により説明される。ファイルシステムに与えられるデータは、ブロック装置に取り付けられたデバイスドライバにより与えられるファイルシステムサポートのレベルに基づいている。これらの構成は、ローカルファイルシステムに対してのみ等価性を有している。

```
/*file system independent-volume params*/
struct vpfsi {
    unsigned long vpi_vid;      /* 32ビットボリューム ID*/
    unsigned long vpi_hDEV;     /*デバイスドライバへのハンドル*/
    unsigned short vpi_bsize;   /*バイトのセクタサイズ*/
    unsigned long vpi_totsec;   /*セクタの全数*/
    unsigned short vpi_trksec;  /*セクタの全数*/
    unsigned short vpi_nhead;   /*セクタの全数*/
    char vpi_text [12] ;        /* ascizボリューム名*/
} ;  /*vpfsi*/
/* file system dependent-volume params*/
struct vpfsd {
        char vpd_work  [36] ; /*作業領域*/
} ;
        /* vpfsd*/
```

上記のように、FS_MOUNT機能は、ボリュームをマウントしかつアンマウントすべく呼び出され、その目的は、FSD がファイルシステムのフォーマ

ットを認識しているか否かを決定すべくボリュームを試験することにある。FS_MOUNTを呼び出すシーケンスは下記の通りである。

```
int far pascal FS_MOUNT(flag, pvpfsi, pvpfsd,
 pBoot)
unsigned short flag;
struct vpfsi far* pvpfsi;
struct vpfsd far* pvpfsd;
unsigned short hVPB;
char far* pBoot;
```

ここで、フラグは、要求された作業を示す。

flag＝0は、ボリュームをマウント又は受け入れるのにFSDが要求されることを示す。

flag＝1は、特定のボリュームが除去されたことをFSDがアドバイスされていることを示す。

flag＝2は、ボリュームがそのドライバから除去されるときに当該ボリュームに割り当てられる全ての内部情報及び当該ボリュームが除去されたことの最終のカーネル管理リファレンスを開放するのにFSDが要求されることを示す。

flag＝3は、FSDに使用するためのフォーマット化の準備において、認識の如何に係わらずボリュームを受け入れることをFSDが要求されることを示す。

他の全てのバリューはリザーブされる。FSDに導かれるバリューは正しいであろう。

pvpfsi－VPBのファイルシステム独立部分を指すポインタ。メディアが、オペレーティングシステムの認識可能なブートセクタを収容している場合には、ファイルされたvpi_vidは、ボリュームについての３２ビット識別子を収容する。メディアがそのようなブートセクタを収容しない場合には、FSDは、メディアに対してユニークなラベルを創出して、該ラベルをファイルされたvpi_vidに配置する。

pvpfsd－VPBのファイルシステム従属部分を指すポインタ。FSDは、必要に応じ情報をこの領域に記憶させることができる。

hVPB－ボリュームへのハンドル。

pBoot －メディアから読み取られたセクタ0を指すポインタ。このポインタは、フラグ＝＝0のときにのみ正当である。ポインタのバッファは、「MUST NOT BE MODIFIED（変更してはならない）」を指示する。ポインタは常に正当であり、フラグ＝＝0である場合にはその正当であることを確認する必要はない。読取りエラーが生じた場合には、バッファはゼロを包含する。

FSDは、もたらされたボリュームを試験し、ボリュームがファイルシステムを認識しているか否かを決定する。ボリュームがファイルシステムを認識している場合には、vpfsi及びvpfsdの適当な部分を充填した後に、ゼロに戻る。vpi_vid及びvpi_textのフィールドは、FSDにより充填される。FSDがオペレーティングシステムのフォーマットブートセクタを有している場合には、FSDは、メディアをラベルからasciizフォームに変換する。vpi_hDevのフィールドは、オペレーティングシステムにより充填される。ボリュームが認識されていない場合には、ドライバは非ゼロ（non-zero）に戻る。

vpi_text及びvpi_vidは、これらのバリューが変化する度毎に、FSDにより更新される。

vpfsdの内容は次の通りである。

FLAG＝0

FSDはFSD_FINDDUPHVPBを発行して、複製VPBが存在しているか否かを決定する。複製VPBが存在している場合には、新しいVPBのfs従属領域が正しくなく、FSDがFS_MOUNTの呼出しから戻った後に、新しいVPBがアンマウントされる。FSDは、古いVPBのfs従属領域を更新する。

複製VPBが存在しない場合には、FSDは、fs従属領域を初期化する。

FLAG＝1

VPBのfs従属部分は、FSDが最後にこの従属部分を変更したものと同じである。

FLAG＝2

VPBのfs従属部分は、FSDが最後にこの従属部分を変更したものと同じである。

メディア認識プロセスの後、FSH_GETVOLPARMM呼出しを用いて、ボリュームパラメータを試験する

ことができる。ボリュームパラメータは、メディア認識プロセスの後に変更すべきではない。

マウントリクエストの間、FSD は、FSH_DOVOLIO を用いることによりメディアの他のセクタを試験し、I/O を行うことができる。不確実メディアの戻りが検出される場合には、FSD は「cleans up（クリナップ）」の状態になってERROR_UNCERTAIN_MEDIAを戻し、新しくインサートされたメディアに関してボリュームマウント論理が再スタートできるようにする。FSD は、付加I/O に使用できるバッファを設ける。

オペレーティングシステムのカーネルは、上記レフカウント(refcount)のカウンタを介してVPB を管理する。全てのボリューム特定目的(volume-specific objects) は、適当なボリュームハンドルでラベリングされ、VPB に対する基準を示す。ボリュームに対する全てのカーネル基準が消滅すると、フラグ＝2でFS_MOUNT が呼び出され、ディスマウントリクエストを表示する。

ボリュームがそのドライブから除去されたことをカーネルが検出し、かつ、依然としてボリュームに対する未解決の基準 (outstanding references)が存在すると、FS_MOUNT はフラグ＝1で呼び出され、FSD が、ボリュームについてのクリーンなデータ（又は、他の再生可能なデータ）を記憶できるようにする。ダーティで再生不可能なデータは、該データがドライブ内にリマウントされるときに、ボリュームに書き込むことができるように保持される。本発明の目的を得る上で、クリーンデータは変化されないデータであり、ダーティなデータは変更されたデータである。

FSD に使用できるようにするためボリュームをフォーマット化すべきときには、オペレーティングシステムのカーネルは、フラグ＝3でFSD のFS_MOUNT 入口を呼び出し、FSD がフォーマット作業の準備を行えるようにする。FSD は、ボリュームがFSD の認識するボリュームでない場合でもボリュームを受け入れる。これは、フォーマットがボリュームに関してファイルシステムを変化させるからである。フォーマット化を完了できない場合（例えば、FSD がCD-ROMのみをサポートする場合）には、作業は失敗するであろう。

幾どのコンピュータシステムのハードウェアは、メディアのカーネルメディア形除去(Kernel-mediated removal of media)ができないので、ボリュームがどのドライブにも存在しないときにアンマウントリクエストが発行されることは確実である。FSH_DOVOLIO は、特定のボリュームに対するI/O を実行する。FSH_DOVOLIO は、リクエストされたI/O に対するデバイスドライバのリクエストパケットをフォーマット化し、デバイスドライバを呼出し、かつFSD に戻る前に、あらゆるエラーをハードエラーダエモンに報告する。ハードエラーダエモンにより表示された全ての再試行(retries) 又はDOSERRORにより表示されたアクションは、FSH_DOVOLIO への呼出しを行っている間に行われる。

次に、FSH_DOVOLIO に対する呼出しフォーマットについて説明する。

```
int far pascal FSH_DOVOLIO (operation, hVPB,
PData, pcSec, iSec)
unsigned short operation;
unsigned short hVPB;
char far* pData;
unsigned short far* pcSec;
unsigned long iSec;
```

ここで、オペレーションビットマスク(operation bit mask) は、実行すべきread/read-bypass/write/write-bypass/verify-after-write/write-through及びノーキャッシュ作業(no-cache operation) を表示する。

Bit 0 ×0001 offは、読取りを表示する。

Bit 0 ×0001 on は、書込みを表示する。

Bit 0 ×0002 offは、no-bypass を表示する。

Bit 0 ×0002 on は、cache bypass を表示する。

Bit 0 ×0004 offは、no verify-after-write 作業を表示する。

Bit 0 ×0004 on は、verify-after-writeを表示する。

Bit 0×0008 offは、ハードエラーダエモンに信号を送られたエラーを表示する。

Bit 0×0008 on は、ハードエラーが直接戻されることを表示する。

Bit 0×0010 offは、I/O が "write-through" でないことを表示する。

Bit 0×0010 on は、I/O が "write-through" であることを表示する。

Bit 0×0020 offは、このI/O に対するデータを変更すべきことを表示する。

Bit 0×0020 on は、このI/O に対するデータを変更すべきではないことを表示する。

リザーブされた他の全てのビットはゼロである。

「cache bypass（キャッシュバイパス）」と、「no cache（のーキャッシュ）」ビットとの間の相違は、リクエストパケットの形式においてデバイスドライバが導かれることである。「cache bypass」では、コマンドコード24、25又は26でパケットが得られ、「no cache」では、システムは、コマンドコード4、8又は9に対するパケットを拡大する。

hVPB　I/Oの資源に対するボリュームハンドル。

pData　ユーザの転送領域(transfer area) の長いアドレス。

pcSec　転送すべきセクタ数を指すポインタ。戻り時には、これは首尾良く転送されたセクタ数である。

iSec　転送の最初のセクタ数。

Returns －作業が失敗した場合には、0以外のエラーコード。

ERROR_PROJECTION_VIOLATION－供給されたaddress/length は正しくない。

ERROR_UNCERTAIN_MEDIA －メディアが変更されているときには、デバイスドライバは信頼性をもって告げることはできない。このことは、FS_MOUNTのコンテクスト内においてのみ生じる。

ERROR_TRANSFER_TOO_LONG －デバイスにとって転送が非常に短い。

FSH_DOVOLIO は、常時、FSD 内で使用することができる。FS_MOUNT呼出しの範囲内で呼び出されるとき、FSH_DOVOLIO は、ボリュームの如何に係わらず、ドライブのボリュームに適用される。しかしながら、FSD がFS_MOUNTの呼出しに戻るまでは、ボリュームの認識が完了しないので、FSD は、ERROR_UNCERTAIN_MEDIA が提されないときには、特別な注意を払わなければならない。これは、ドライブにおけるメディアを識別するのに、メディアが不確実な試みを行ったことを表示する。また、FSD が認識を試みたボリュームが除去されたことを表示するようにしてもよい。この場合、FSD は、FS_MOUNTの呼出し時に導かれたhVPBに取り付けられたあらゆる資源を解放し、かつERROR_UNCERTAIN_MEDIAをFS_MOUNT呼出しに戻す。これにより、マウントプロセスを再スタートさせるように、ボリュームトラッキング論理が仕向けられる。

FSDsは、FSH_DOVOLIO2を呼び出して、I/O 作業とは独立して、デバイスドライバの作業を制御する。このルーチンは、IOCTL 作業に対するボリューム管理をサポートしている。FSD に戻る前に、全てのエラーがハードエラーダエモンに報告される。ハードエラーダエモンにより表示されるすべての再試行又はDOSERRORにより表示されるアクションは、FSH_DOVOLIO2への呼出し内に行われる。

```
int far pascal FSH_DOVOLIO2 (hDev, sfn, cat,
func, pParm, cbParm, pData, cbData)
unsigned long hDev;
unsigned short sfn;
unsigned short cat;
unsigned short func;
char far* pParm;
unsigned short cbParm;
char far* pData;
unsigned short cbData;
```

ここで、

hDev－VPB から得られたデバイスハンドル。

sfn －FSH_DEVIOCTL呼出しを引き起こしたオープンインスタンス(open instance) からのシステムファイルの数。このフィールドは、変化されない状態で、sfi selfsfn のフィールドから導か

れるべきである。どのオープンインスタンスもこの呼出しに一致しないときには、このフィールドは、0×FFFFに設定される。

cat －実行すべきIOCTL のカテゴリ。

func－IOCTL のカテゴリ内での機能。

pParm －パラメータ領域へのロングアドレス。

cbParm－パラメータ領域の長さ。

pData －データ領域へのロングアドレス。

cbData－データ領域の長さ。

Returns －エラーが検出されないときには0以外のエラーコード。

供給された機能が、ここに述べているシステムとの互換性をもたない場合には、ERROR_INVALID_FUNCTIONが呼び出される。メディアが不確実になるときにはいつでも、新しいVPB が割り当てられる（メディアが変更されていないことがもはや確かではないことを、デバイスドライバは認識している）。このVPB は、FS_MOUNT呼出しが戻るまで、（メディアの再インサートにより）前に割り当てられたVPB と共に崩壊することはない。しかしながら、前のVPB は、メディア（このメディアは、該メディアが除去されている間に書き込むことができる）から更新されなくてはならない幾つかのキャッシュデータをもつことができる。古いVPB についてのキャッシュ情報（cached information）を更新するため、FSH_FINDDUPVPB は、ボリュームの、この前に生じたことをFSD が見出すことを可能にする。このボリュームについて別の古いVPB が存在しない場合には、新しく創出されたVPB がアンマウントされる。

FSH_FINDDUPNVPB についての呼出しフォーマットは次の通りである。

```
int far pascal FSH FINDDUPNVPB (hVPB. phVPB)
unsigned short hVPB;
unsigned short far* phVPB;
```

ここで、

hVPB－見出すべきボリュームに対するハンドル。

phVPB －マッチングボリュームのハンドルをどこに記憶させるかを指すポインタ。

Returns －マッチングVPB が見出されないときには0以外のエラーコード。

ERROR_NO_ITEMS－マッチングhVPBは存在しない。

FSH_GETVOLPARMは、FSD が、VPB からファイルシステムの独立及び従属データを検索することを可能にする。FSルートはVPB ハンドル内を通るので、ここのPSDsは、適合部分を指すポインタ内にハンドルをマッピングする。FSH_GETVOLPARMについての呼出しシーケンスは次の通りである。

```
void far pascal FSH GETVOLPARM (hVPB, ppVPB-
fsi, ppVPBfsd)
unsigned short hVPB;
struct vpfsi far* far* ppVPBfsi;
struct vpfsi far* far* ppVPBfsd;
```

ここで、

hVPB－関心のあるボリュームハンドル。

ppVPBfsi－ファイルシステムの独立データを記憶させるべくポインタが指す位置。

ppVPBfsd－ファイルシステムの従属データを記憶させるべくポインタが指す位置。

Returns －なし。

FSD_Volumeのマッピングはダイナミックであり、かつFSD-DD連結は、FSD 及びDDの独立方法により、オペレーティングシステムのカーネルを介して行われるので、あらゆるFSD は、DDs がこのFSD からローディングされたボリュームを含むあらゆるボリュームにアクセスできる。ボリュームは、除去可能なメディアの特定のピース（片）、又は区分できる任意のメディアの任意の区分にマッピングするので、多数のFSDsが特定のハードディスク又は他のメディアにアクセスできるようにしてもよい。

ボリュームファイル作業は、2つのカテゴリ、すなわち、ネームベース作業（named-based operations）及びハンドルベース作業（handle-based operations）に分けることができる。ネームベース作業は一般にユーザによって開始され、システムIOCTLがファイルについてネームド作業を行うことをユーザがシステムIOCTLに命令（instruct）する。ハンドルベース作業は、一般に、システムのバックグラウンド作業中に開始される。通常、

ハンドルベース作業は、ネームベース作業の後に行われる。

第８図に示すように、コンピュータシステム１００がネームベース作業を行っているときに、ルーチン８００が呼び出される。ネーム作業は、文字ネームにより指示された作業である。すなわち、この作業は、ファイル又はディレクトリのネームにより特定化される。「Open file "xxx"」は、ネームベース作業の一例である。プロセス８０２は、ネームをパーシング（parse）すべく呼び出されて、３つの変数すなわち、PathNameType、TCBThisVPB 及びTCBThisFCBを戻す。このプロセス８０２については、第９図に関連して詳細に説明する。（注、h はハンドルをいい、TCB は、TCBThisVPB が現に関心をもっているVPB へのハンドルでありかつTCBThisFCBが関心のあるファイルシステムを指すポインタである。スレッド制御ブロック（thread control block）をいう。）次に、項目８０４は、プロセス８０２により戻された変数PathType、TcbThisVPB 及びTCBThisFCBに基づく適当な機能に制御をルーチング（route）する。UNC FSD が呼び出されるユニバーサルネーミングコンベンション（Universal Naming Convention、「UNC」）のグローバルネットワークを表示する「＼＼」で経路（path）が始まるときには、項目８０６を通る制御が行われる。ローカル装置が表示される場合には、制御が項目８０８に導かれ、カーネル内でリクエストが処理される。疑似装置又は遠隔ファイルが表示されるときには、制御は項目８１０に導かれて、疑似装置又は遠隔ファイルが取り付けられた遠隔FSD に対するリクエストをルーチングする。ネームドパイプ（named pipe）が検出された場合には、制御は項目８１２に導かれて、カーネル無いでローカルネームドパイプコードを呼び出す。ローカルファイルが表示される場合には、制御は項目８１４に導かれる。この項目８１４は、項目８１６におけるFSHDOVOLIOを呼び出すことによりボリュームへの読取り及び書込みを行うFSD におけるFSD ワーカである。FSHDOVOLIO については、第１１図に関連して更に詳細に説明する。

次に、第９図を参照して、パーシングプロセス８０２を説明する。呼出しがあると、項目９０２は、現在のドライブ、現在のディレクトリ及びネーム自体に基づいて、関心のあるネームをカノニカルフォーム（canonical form）に変形する。次に、変数TCBThisFSC、TCBThisVPB及びpathnametype が下記のようにして決定される。デシジョン９０４は、ユーザのネームが「＼＼」で始まっているか否かを決定して、UNC ネームが表示されているか否かを決定する。UNC ネームが表示されていれば、制御が項目９０５に導かれ、ここで、変数pathtype、TcbThisVPB 及び TCBThisFCB が初期化され、ユーザのネームを適当な位置にルーチングする。UNC ネームが表示されていなければ、デシジョン９０６は、関心をもつネームがカーネルにより維持された、装置のネームリストに載ったネームであるか否かを決定する。装置のネームリストに載ったネームであるときには、デシジョン９０８は、それが疑似文字装置であるか否かを決定する。疑似文字装置であれば、項目９１０は表示のように変数を設定する。疑似文字装置でないときは、制御は項目９１２に導かれ、表示のように変数を設定する。

デシジョン９１４は、ネームの初めにおける「＼pipe＼」を探すことにより、そのネームが、ネームドパイプであるか否かを決定する。そのネームがネームドパイプであるときは、項目９１６は、表示のように変数を設定する。ネームドパイプでないときは、デシジョン９１８は、そのネームがローカルドライブ又は遠隔ドライブのパスネーム（pathname）を表示しているか否かを決定する。遠隔ドライブが表示されているときは、制御は項目９２０に導かれ、ここでは、表示のように変数PathType、TcbThisVPB 及びTCBThisFCBを設定する。遠隔ドライブが表示されていないときは、制御は項目９２２に導かれ、ここでは、どのボリュームから適当なデータを読み取るかが呼び出される。WhatVolumeが戻ると、制御は項目９２４に導かれ、ここでは、表示のように変数PathType、

TcbThisVPB 及びTCBThisPCBが設定される。

第１０図を参照すると、プロセス1006が呼び出され、ハンドルベース作業が行われる。呼出し時に、項目1002はSFT 入口を検索する。SFT 入口及びハンドルは、両方共DosOpen により設定される。次いで、TCBThisPSCが表示のように設定される。次に、項目1004において、FSC が指示するファイルシステムの適合FSH ワーカが呼び出される。次いで、項目1006は、項目1016を呼び出し、必要に応じて項目1016を呼び出すことにより、呼出し者すなわちコーラー(caller)によりリクエストされたあらゆるI/O を実行する。

第１１図には、FSH Do Vol IO が示されている。項目1102において呼出しがなされると、hVPBが使用されて、ドライブ並びに関心のあるボリュームにどのようなボリュームがあるかを決定する。次に、デシジョン1104は、ドライブにおけるボリュームが関心のあるボリュームであるか否かを決定する。関心のあるボリュームであるときは、項目1106が呼び出されて、デバイスドライバが呼び出されかつ特定のパラメータでI/O を実行する。次にデシジョン1108は、作業注にメディアが不確実に移行したか否かを決定する。不確実に移行していないときには、プロセスは項目1114に戻る。デシジョン1108が、メディアは不確実ではないことを決定するときには、制御は項目1112に導かれ、ここでは、WhatVolume が呼び出されてメディアが確実なものにされる。次いで、制御はデシジョン1104に戻る。ドライブにおけるボリュームが関心のあるボリュームに一致しないときには、項目1110が呼び出されて、HardError が呼び出され、ドライブに正しいドライブを置くことをユーザに知らせる。次に制御は上記項目1112に導かれる。

付録（Appendix） IIからVは、設置可能なファイルシステムの資源の一例としてここに包含される。ここで、

付録Ｂは、本発明の教示に従ってサポートすることが期待されるファイルシステムの移出インターフェース（exported interfaces）のリストである。

付録Ｃは、ファイルシステムを用いることができるカーネルにより移出されるインターフェースのリストである。

付録Ｄは、本発明に従って構成された設置可能なファイルシステムの一例の資源コードである。

付録Ｅは、付録ＤのFSD を作るべくOS/2により用いられる定義ファイル(definitions file)のリストである。

以上、設置可能なファイルシステムにおいてダイナミックボリュームのトラッキングを行う方法及びシステムについて説明した。本発明の教示によれば、リンクされたリストにおいて編成される複数のファイルシステムドライバを、コンピュータシステムに設けることができる。コンピュータシステムには更に、省略時ファイルシステムドライバを設けることができる。コンピュータシステムに新しいボリュームが与えられたとき、又はメディアが不確実になったときにはいつでも、コンピュータシステムは、ファイルシステムドライバがボリュームを受け入れるまで、自動的かつダイナミックにファイルシステムドライバを呼び出すようになっている。どのファイルシステムドライバもボリュームを受け入れない場合には、省略時ファイルシステムがマウントされる。従って、当業者には、本発明の他の用法及び変更が可能であり、そのようなあらゆる用法及び変更は本発明の範囲及び精神に含まれるものである。

4.図面の簡単な説明

第１Ａ図は、本発明の原理に従って構成されたコンピュータシステムのブロックダイアグラムである。

第１Ｂ図は、第１Ａ図のコンピュータシステムの作業及びファイルシステムアーキテクチュアを示すブロックダイアグラムである。

第２Ａ図は、MS-DOSオペレーティングシステムのファイルシステム構成の詳細を示すブロックダイアグラムである。

第２Ｂ図は、本発明の設置可能なファイルシステムのファイルシステム構成の詳細を示すブロックダイアグラムである。

第３図は、第２Ｂ図のファイルシステムのより詳細なブロックダイアグラムである。

第４図は、FAT ファイルシステムのディスクフォーマットを示すブロックダイアグラムである。

第５Ａ図～第５Ｈ図は、本発明に使用できるように構成された設置可能なファイルシステムのディスクフォーマットを示すブロックダイアグラムである。

第６図は、本発明のマウントプロセスの全作業の詳細を示すフローチャートである。

第７図は、本発明の設置可能なファイルシステムの構成を示すブロックダイアグラムである。

第８図は、本発明の原理に従がうネームベース作業の実行の詳細を示すフローチャートである。

第９図は、ネームベース作業プロセスにより呼び出されるパーシングプロセスを示すフローチャートである。

第１０図は、本発明の原理に従がうハンドルベース作業の実行を示すフローチャートである。

第１１図は、第８図及び第１０図に関連して説明したプロセスにより呼び出されるFSH_DoVolIo プロセスを示すフローチャートである。

１００…コンピュータシステム、
１０２…マイクロプロセッサ、
１０４…ランダムアクセスメモリ、
１０６…リードオンリメモリ、
１０８…マウス、
１１０…キーボード、
１１２…ディスプレイ、
１１４…プリンタ、
１１６…フロッピディスクドライブ、
１２０…ハードディスクドライブ、
１２２…CD-ROMドライブ、
１２４…テープドライブ、
１２６…ネットワーク。

図面の浄書(内容に変更なし)

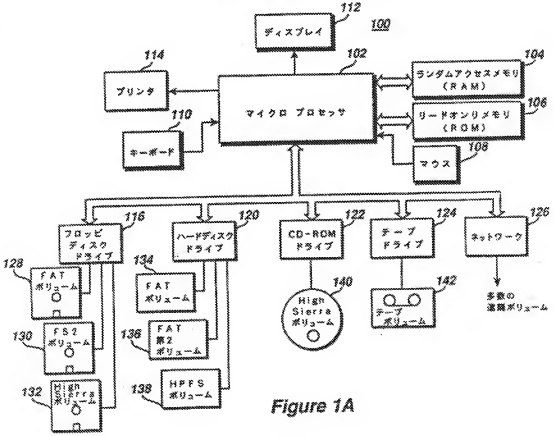

Figure 1A

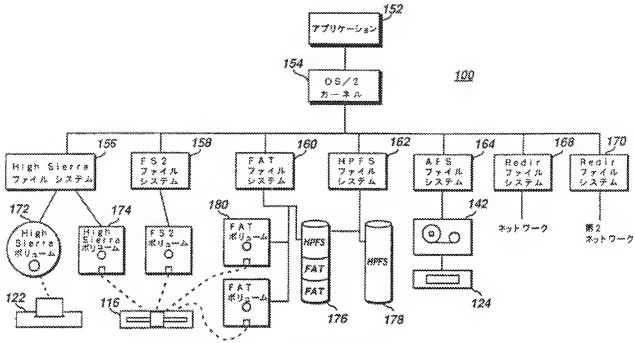

Figure 1B

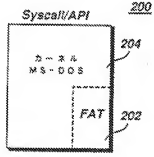

Figure 2A

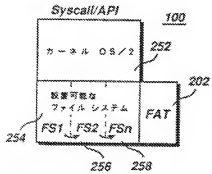

Figure 2B

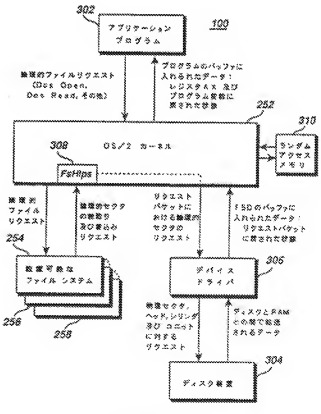

Figure 3

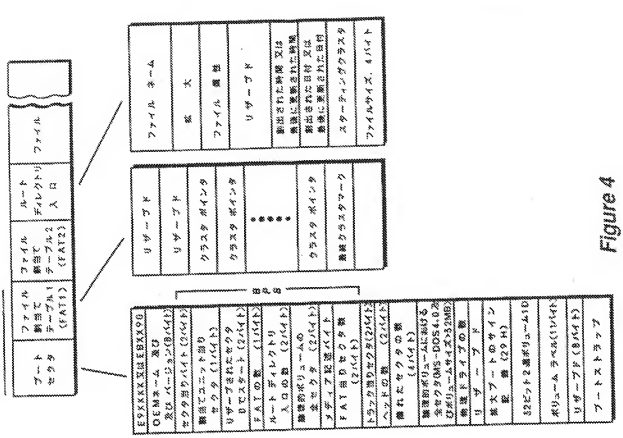

Figure 4

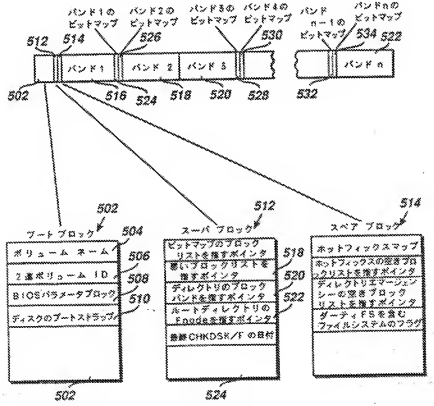

Figure 5A

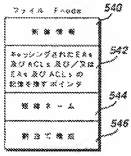

Figure 5B

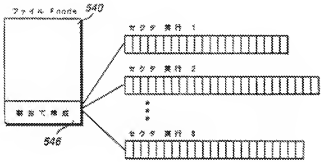

Figure 5C

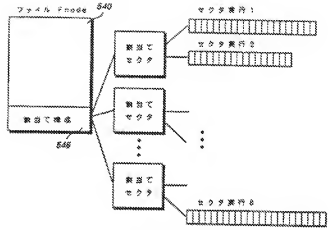

Figure 5D

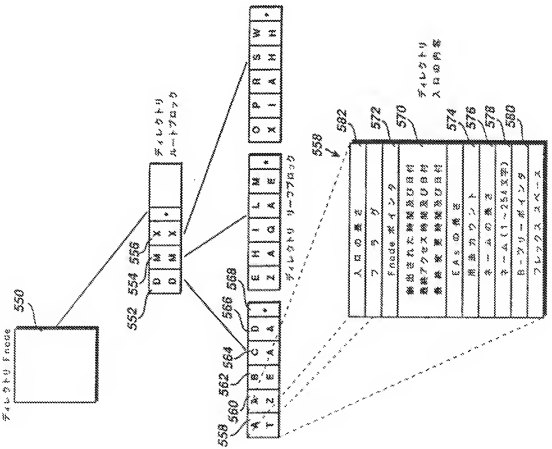

Figure 5E

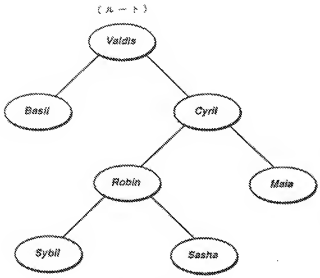

Figure 5F

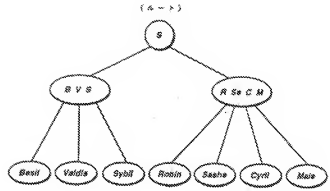

Figure 5G

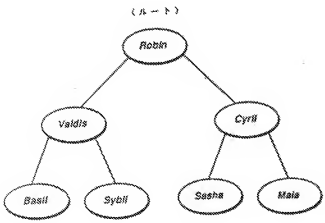

Figure 5H

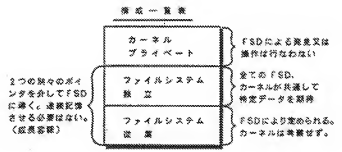

Figure 7

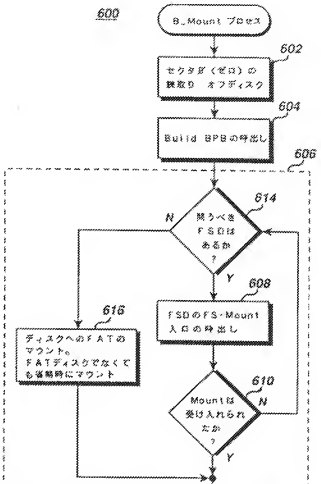

Figure 6

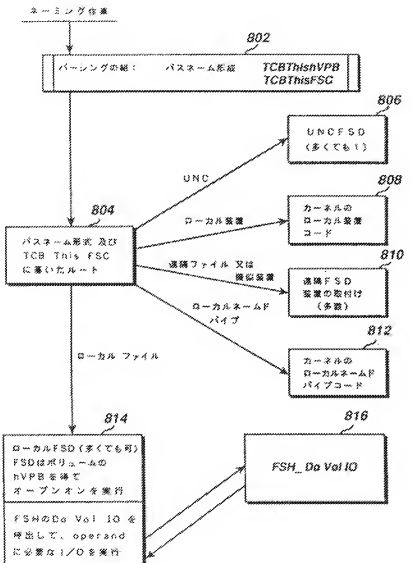

Figure 8

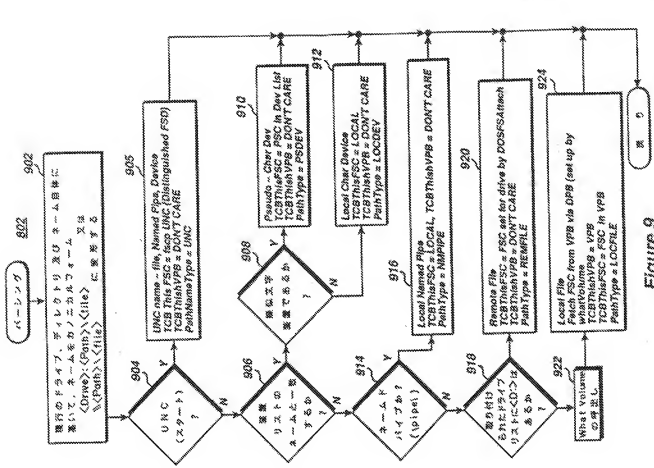

Figure 9

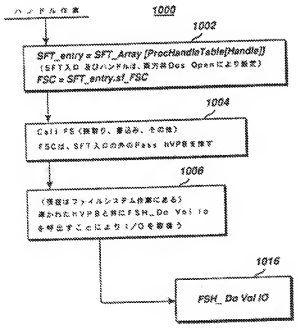

Figure 10

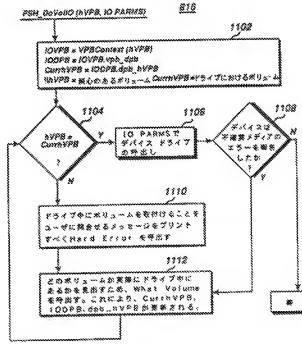

Figure 11

第1頁の続き

⑫発 明 者	マーク ジエイ ズビコースキ	アメリカ合衆国 ワシントン州 98072 ウツデインヴィル ノースイースト ワンハンドレッドアンドセブンティエイス プレイス 15817
⑫発 明 者	ジエイムズ ジー レツウイン	アメリカ合衆国 ワシントン州 98033 カークランド ノースイースト ワンハンドレッドアンドフオース ストリート 11428
⑫発 明 者	ラジエン ジヤヤンテイラル シヤー	アメリカ合衆国 ワシントン州 98005 ベルヴイユー ワンハンドレッドアンドトウエンテイサード アベニユー ノースイースト 517

手 続 補 正 書（方式）

| 2.12.27
平成 年 月 日

特許庁長官 植 松 敏 殿

1.事件の表示 平成2年特許願第227995号

2.発明の名称 記憶可能なファイルシステムにおいてダイナミックボリュームトラッキングを行う方法及び装置

3.補正をする者
事件との関係 出 願 人

名 称 マイクロソフト コーポレーション

4.代 理 人
住 所 東京都千代田区丸の内3丁目3番1号 電話 (代) 211-8741
氏 名 (7955) 弁理士 中 村

5.補正命令の日付 平成2年11月27日

6.補正の対象 願書の特許出願人の欄 代理権を証明する書面 全 図 面

7.補正の内容 別紙のとおり
願書に最初に添付した図面の内容 (内容に変更なし)

特許庁 2.12.27